

ISE 4 In-Depth Tutorial

HDL-Based Designs

Schematic-Based Designs

Behavioral Simulation

Design Implementation

Timing Simulation

iMPACT Tutorial



The Xilinx logo shown above is a registered trademark of Xilinx, Inc.

CoolRunner, RocketChips, RocketIP, Spartan, StateBENCH, StateCAD, Virtex, XACT, XILINX, XC2064, XC3090, XC4005, and XC5210 are registered trademarks of Xilinx, Inc.



The shadow X shown above is a trademark of Xilinx, Inc.

ACE Controller, ACE Flash, A.K.A. Speed, Alliance Series, AllianceCORE, Benchner, ChipScope, Configurable Logic Cell, CORE Generator, CoreLINX, Dual Block, EZTag, Fast CLK, Fast CONNECT, Fast FLASH, FastMap, Fast Zero Power, Foundation, Gigabit Speeds...and Beyond!, HardWire, HDL Bencher, IRL, J Drive, JBits, LCA, LogiBLOX, Logic Cell, LogiCORE, LogicProfessor, MicroBlaze, MicroVia, MultiLINX, NanoBlaze, PicoBlaze, PLUSASM, PowerGuide, PowerMaze, QPro, Real-PCI, Rocket I/O, Select I/O, SelectRAM, SelectRAM+, Silicon Xpresso, Smartguide, Smart-IP, SmartSearch, SMARTswitch, System ACE, Testbench In A Minute, TrueMap, UIM, VectorMaze, VersaBlock, VersaRing, Wave Table, WebFITTER, WebPACK, WebPOWERED, XABEL, XACTstep Advanced, XACTstep Foundry, XACT-Floorplanner, XACT-Performance, XAM, XAPP, X-BLOX +, XChecker, XDM, XEPLD, Xilinx Foundation Series, Xilinx XDTV, Xinfo, XSI, XtremeDSP, all XC designated products, and ZERO+ are trademarks of Xilinx, Inc. The Programmable Logic Company is a service mark of Xilinx, Inc.

All other trademarks are the property of their respective owners.

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described or shown herein; nor does it convey any license under its patents, copyrights, or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. will not assume responsibility for the use of any circuitry described herein other than circuitry entirely embodied in its products. Xilinx, Inc. devices and products are protected under one or more of the following U.S. Patents: 4,642,487; 4,695,740; 4,706,216; 4,713,557; 4,746,822; 4,750,155; 4,758,985; 4,820,937; 4,821,233; 4,835,418; 4,855,619; 4,855,669; 4,902,910; 4,940,909; 4,967,107; 5,012,135; 5,023,606; 5,028,821; 5,047,710; 5,068,603; 5,140,193; 5,148,390; 5,155,432; 5,166,858; 5,224,056; 5,243,238; 5,245,277; 5,267,187; 5,291,079; 5,295,090; 5,302,866; 5,319,252; 5,319,254; 5,321,704; 5,329,174; 5,329,181; 5,331,220; 5,331,226; 5,332,929; 5,337,255; 5,343,406; 5,349,248; 5,349,249; 5,349,250; 5,349,691; 5,355,035; 5,357,153; 5,360,747; 5,361,229; 5,362,999; 5,365,125; 5,367,207; 5,386,154; 5,394,104; 5,397,943; 5,399,924; 5,399,925; 5,406,133; 5,410,189; 5,410,194; 5,414,377; 5,422,833; 5,426,378; 5,426,379; 5,430,687; 5,432,719; 5,448,181; 5,448,493; 5,450,021; 5,450,022; 5,453,706; 5,455,525; 5,466,117; 5,469,003; 5,475,253; 5,477,414; 5,481,206; 5,483,478; 5,486,707; 5,486,776; 5,488,316; 5,489,858; 5,489,866; 5,491,353; 5,495,196; 5,497,108; 5,498,979; 5,498,989; 5,499,192; 5,500,608; 5,500,609; 5,502,000; 5,502,440; 5,504,439; 5,504,440; 5,506,518; 5,506,523; 5,506,878; 5,513,124; 5,517,135; 5,521,835; 5,521,837; 5,523,963; 5,523,971; 5,524,097; 5,526,322; 5,528,169; 5,528,176; 5,530,378; 5,530,384; 5,546,018; 5,550,839; 5,550,843; 5,552,722; 5,553,001; 5,559,751; 5,561,367; 5,561,629; 5,561,631; 5,563,527; 5,563,528; 5,563,529; 5,563,827; 5,565,792; 5,566,123; 5,570,051; 5,570,059; 5,574,634; 5,574,655; 5,578,946; 5,581,198; 5,581,199; 5,581,738; 5,583,450; 5,583,452; 5,592,105; 5,594,367; 5,598,424; 5,600,263; 5,600,264; 5,600,271; 5,600,597; 5,608,342; 5,610,536; 5,610,790; 5,610,829; 5,612,633; 5,614,844; 5,617,021; 5,617,041; 5,617,327; 5,617,573; 5,623,387; 5,627,480; 5,629,637; 5,629,886; 5,631,577; 5,631,583; 5,635,851; 5,636,368; 5,640,106; 5,642,058; 5,646,545; 5,646,547; 5,646,564; 5,646,903; 5,648,732; 5,648,913; 5,650,672; 5,650,946; 5,652,904; 5,654,631; 5,654,665; 5,656,950; 5,657,290; 5,659,484; 5,661,660; 5,661,685; 5,668,495; 5,670,896; 5,670,897; 5,672,966; 5,673,198; 5,675,262; 5,675,270; 5,675,589; 5,677,638; 5,682,107; 5,684,413; 5,689,133; 5,689,516; 5,691,907; 5,691,912; 5,694,047; 5,694,055; 5,694,056; 5,694,399; 5,696,454; 5,701,091; 5,701,441; 5,703,759; 5,705,932; 5,705,938; 5,708,597; 5,712,579; 5,714,890; 5,715,197; 5,717,340; 5,719,506; 5,719,507; 5,724,276; 5,726,484; 5,726,584; 5,734,866; 5,734,868; 5,737,234; 5,737,235; 5,737,631; 5,742,178; 5,742,179; 5,742,531; 5,744,974; 5,744,979; 5,744,981; 5,744,995; 5,748,942;

5,748,979; 5,752,006; 5,752,035; 5,754,459; 5,758,192; 5,760,603; 5,760,604; 5,760,607; 5,761,483; 5,764,076; 5,764,534; 5,764,564; 5,768,179; 5,770,951; 5,773,993; 5,778,439; 5,781,756; 5,784,313; 5,784,577; 5,786,240; 5,787,007; 5,789,938; 5,790,479; 5,790,882; 5,795,068; 5,796,269; 5,798,656; 5,801,546; 5,801,547; 5,801,548; 5,808,479; 5,811,985; 5,815,004; 5,815,016; 5,815,404; 5,815,405; 5,818,255; 5,818,730; 5,821,772; 5,821,774; 5,825,202; 5,825,662; 5,825,787; 5,828,230; 5,828,231; 5,828,236; 5,828,608; 5,831,448; 5,831,460; 5,831,845; 5,831,907; 5,835,402; 5,838,167; 5,838,901; 5,838,954; 5,841,296; 5,841,867; 5,844,422; 5,844,424; 5,844,829; 5,844,844; 5,847,577; 5,847,579; 5,847,580; 5,847,993; 5,852,323; 5,861,761; 5,862,082; 5,867,396; 5,870,309; 5,870,327; 5,870,586; 5,874,834; 5,875,111; 5,877,632; 5,877,979; 5,880,492; 5,880,598; 5,880,620; 5,883,525; 5,883,852; 5,886,538; 5,889,411; 5,889,412; 5,889,413; 5,889,701; 5,892,681; 5,892,961; 5,894,420; 5,896,047; 5,896,329; 5,898,319; 5,898,320; 5,898,602; 5,898,618; 5,898,893; 5,907,245; 5,907,248; 5,909,125; 5,909,453; 5,910,732; 5,912,937; 5,914,514; 5,914,616; 5,920,201; 5,920,202; 5,920,223; 5,923,185; 5,923,602; 5,923,614; 5,928,338; 5,931,962; 5,933,023; 5,933,025; 5,933,369; 5,936,415; 5,936,424; 5,939,930; 5,940,606; 5,942,913; 5,944,813; 5,945,837; 5,946,478; 5,949,690; 5,949,712; 5,949,983; 5,949,987; 5,952,839; 5,952,846; 5,955,888; 5,956,748; 5,958,026; 5,959,821; 5,959,881; 5,959,885; 5,961,576; 5,962,881; 5,963,048; 5,963,050; 5,969,539; 5,969,543; 5,970,142; 5,970,372; 5,971,595; 5,973,506; 5,978,260; 5,986,958; 5,990,704; 5,991,523; 5,991,788; 5,991,880; 5,991,908; 5,995,419; 5,995,744; 5,995,988; 5,999,014; 5,999,025; 6,002,268; 6,002,282; 6,002,991; 6,005,423; 6,005,829; 6,008,666; 6,011,407; 6,011,740; 6,016,063; 6,018,250; 6,018,624; 6,020,633; 6,020,756; 6,020,757; 6,020,776; 6,021,423; 6,023,564; 6,023,565; 6,025,736; 6,026,481; 6,028,445; 6,028,450; 6,033,938; 6,034,542; 6,034,548; 6,034,557; 6,035,106; 6,037,800; 6,038,386; 6,041,340; 6,043,692; 6,044,012; 6,044,025; 6,046,603; 6,047,115; 6,049,222; 6,049,227; 6,051,992; 6,054,871; 6,055,205; 6,057,589; 6,057,704; 6,057,708; 6,061,417; 6,061,418; 6,067,508; 6,069,488; 6,069,489; 6,069,490; 6,069,849; 6,070,260; 6,071,314; 6,072,348; 6,073,154; 6,074,432; 6,075,418; 6,078,201; 6,078,209; 6,078,528; 6,078,735; 6,078,736; 6,081,914; 6,084,429; 6,086,629; 6,086,631; 6,091,262; 6,091,263; 6,091,892; 6,094,063; 6,094,065; 6,094,385; 6,097,210; 6,097,238; 6,099,583; 6,100,705; 6,101,132; 6,101,143; 6,104,211; 6,105,105; 6,107,821; 6,107,826; 6,107,827; 6,112,322; 6,114,843; 6,118,286; 6,118,298; 6,118,300; 6,118,324; 6,118,869; 6,118,938; 6,120,549; 6,120,551; 6,121,795; 6,124,724; 6,124,731; 6,130,550; 6,133,751; 6,134,191; 6,134,517; 6,137,307; 6,137,714; 6,144,220; 6,144,225; 6,144,262; 6,144,933; 6,150,838; 6,150,839; 6,150,863; 6,154,048; 6,154,049; 6,154,052; 6,154,053; 6,157,209; 6,157,211; 6,157,213; 6,160,418; 6,160,431; 6,163,167; 6,167,001; 6,167,416; 6,167,545; 6,167,558; 6,167,560; 6,172,518; 6,172,519; 6,172,520; 6,173,241; 6,175,246; 6,175,530; 6,177,819; 6,177,830; 6,181,158; 6,181,164; 6,184,708; 6,184,709; 6,184,712; 6,185,724; 6,188,091; 6,191,610; 6,191,613; 6,191,614; 6,192,436; 6,195,774; 6,199,192; 6,201,406; 6,201,410; 6,201,411; and 6,202,106; Re. 34,363, Re. 34,444, and Re. 34,808. Other U.S. and foreign patents pending. Xilinx, Inc. does not represent that devices shown or products described herein are free from patent infringement or from any other third party right. Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made. Xilinx, Inc. will not assume any liability for the accuracy or correctness of any engineering or software support or assistance provided to a user.

Xilinx products are not intended for use in life support appliances, devices, or systems. Use of a Xilinx product in such applications without the written consent of the appropriate Xilinx officer is prohibited.

Copyright 1991-2001 Xilinx, Inc. All Rights Reserved.

About This Manual

About the In-Depth Tutorial

This tutorial give a description of the features and additions to Xilinx's newest product—ISE 4. The primary focus of this tutorial is to show the relationship between the design entry tools, Xilinx and third-party tools, and the design implementation tools.

This guide is a learning tool for designers who are unfamiliar with the features of the ISE software or those wanting to refresh their skills and knowledge.

You may choose to follow one of four tutorial flows available in this document. For information about the tutorial flows, see “[Tutorial Flows](#).”

Additional Resources

For additional information, go to <http://support.xilinx.com>. The following table lists some of the resources you can access from this page. You can also directly access some of these resources using the provided URLs.

Resource	Description/URL
Tutorial	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://support.xilinx.com/support/techsup/tutorials/index.htm
Answers Database	Current listing of solution records for the Xilinx software tools Search this database using the search function at http://support.xilinx.com/support/searchtd.htm
Application Notes	Descriptions of device-specific design techniques and approaches http://support.xilinx.com/apps/appsweb.htm

Resource	Description/URL
Forums	Discussion groups and chat rooms for Xilinx software users http://toolbox.xilinx.com/cgi-bin/forum
Data Book	Pages from <i>The Programmable Logic Data Book</i> , which describe device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://support.xilinx.com/partinfo/databook.htm
Xcell Journals	Quarterly journals for Xilinx programmable logic users http://support.xilinx.com/xcell/xcell.htm
Tech Tips	Latest news, design tips, and patch information on the Xilinx design environment http://support.xilinx.com/support/techsup/journals/index.htm

Tutorial Contents

This guide covers the following topics.

- **Chapter 1, “HDL-Based Design,”** guides you through a typical HDL-based design procedure using a design of a runner’s stopwatch called “Watch”.
- **Chapter 2, “Schematic-Based Design,”** explains many different facets of a schematic-based ISE design flow using a design of a runner’s stopwatch called “Watch”. This chapter also shows how to use ISE accessories such as StateCad, Project Navigator, LogiBLOX, and HDL Editor.
- **Chapter 3, “Behavioral Simulation,”** explains how to use the Logic Simulator to simulate a design before design implementation to verify that the logic that you have created is correct.
- **Chapter 4, “Design Implementation,”** describes how to Translate, Map, Place, Route, (Fit for CPLDs) and generate a Bit file for designs.

-
- **Chapter 5, “Timing Simulation,”** explains how to perform a timing simulation using the block and routing delay information from the routed design to give an accurate assessment of the behavior of the circuit under worst-case conditions.
 - **Chapter 6, “iMPACT Tutorial”** describes Xilinx's next generation device programming tool, iMPACT.

Tutorial Flows

This document contains four tutorial flows. In this section, the four tutorial flows are outlined and briefly described, in order to help you determine which sequence of chapters applies to your needs. The tutorial flows include:

- HDL Design Flow
- Schematic Design Flow
- Implementation-only Flow
- IMPACT Flow

HDL Design Flow

The HDL Design flow is as follows:

- **Chapter 1, “HDL-Based Design.”**
- **Chapter 3, “Behavioral Simulation.”**
Note that behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 4, “Design Implementation.”**
- **Chapter 5, “Timing Simulation.”**
Note that timing simulation is optional; however, it is strongly recommended.
- **Chapter 6, “iMPACT Tutorial.”**
IMPACT is optional. For customers who want to download their design onto an FPGA/CPLD.

Schematic Design Flow

The Schematic Design flow is as follows:

- **Chapter 2, “Schematic-Based Design.”**
- **Chapter 3, “Behavioral Simulation.”**
Note that behavioral simulation is optional; however, it is strongly recommended in this tutorial flow.
- **Chapter 4, “Design Implementation.”**
- **Chapter 5, “Timing Simulation.”**
Note that timing simulation is optional; however, it is strongly recommended.
- **Chapter 6, “iMPACT Tutorial.”**
IMPACT is optional. For customers who want to download their design to an FPGA/CPLD.

Implementation-only Flow

The Implementation-only flow is as follows:

- **Chapter 4, “Design Implementation.”**
- **Chapter 5, “Timing Simulation.”**
Note that timing simulation is optional; however, it is strongly recommended.
- **Chapter 6, “iMPACT Tutorial.”**
IMPACT is optional. For customers who want to download their design to an FPGA/CPLD.

IMPACT Flow

The IMPACT flow is as follows:

- **Chapter 6, “iMPACT Tutorial.”**
For customers who want to download their design to an FPGA/CPLD.

Conventions

This manual uses the following typographical and online document conventions. An example illustrates each typographical convention.

Typographical

The following conventions are used for all documents.

- `Courier` font indicates messages, prompts, and program files that the system displays.

`speed grade: -100`

- **Courier bold** indicates literal commands that you enter in a syntactical statement. However, braces “{ }” in Courier bold are not literal and square brackets “[]” in Courier bold are literal only in the case of bus specifications, such as bus [7:0].

rpt_del_net=

Courier bold also indicates commands that you select from a menu.

File → Open

- *Italic font* denotes the following items.
 - Variables in a syntax statement for which you must supply values

edif2ngd *design_name*

- References to other manuals

See the *Development System Reference Guide* for more information.

- Emphasis in text

If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.

- Square brackets “[]” indicate an optional entry or parameter. However, in bus specifications, such as bus [7:0], they are required.

edif2ngd [*option_name*] *design_name*

- Braces “{ }” enclose a list of items from which you must choose one or more.

lowpwr = {**on** | **off**}

- A vertical bar “|” separates items in a list of choices.

lowpwr = {**on** | **off**}

- A vertical ellipsis indicates repetitive material that has been omitted.

```
IOB #1: Name = QOUT'  
IOB #2: Name = CLKIN'  
.  
.  
.
```

- A horizontal ellipsis “. . .” indicates that an item can be repeated one or more times.

allow block *block_name loc1 loc2 . . . locn*;

- “Right-click” means click the right mouse button. Unless specified, all other mouse operations are performed with the left mouse button.
- Throughout this tutorial, file names, project names, and directory names (paths) are specified in lower case
- The design used in this tutorial is referred to as Watch.

Online Document

The following conventions are used for online documents.

- [Blue text](#) indicates an intrabook link, which is a cross-reference within a book. Click the blue text to open the specified cross-reference.

-
- [Blue underlined text](#) indicates a Web site. Click the link to open the specified Web site. You must have a Web browser and internet connection to use this feature.

Contents

About This Manual

About the In-Depth Tutorial	v
Additional Resources	v
Tutorial Contents	vi
Tutorial Flows	vii
HDL Design Flow	vii
Schematic Design Flow	viii
Implementation-only Flow	viii
IMPACT Flow	viii

Conventions

Typographical	ix
Online Document	x

Chapter 1 HDL-Based Design

Getting Started	1-2
Required Software	1-2
Installing the Tutorial	1-2
Tutorial Project Directories and Files	1-2
Starting the ISE Software	1-3
VHDL or Verilog?	1-4
Overview of Project Navigator	1-4
Sources in Project Window	1-6
Module View	1-6
Snapshot View	1-6
Library View	1-6
Processes for Current Source Window	1-7
Process View	1-7
Console Window	1-7
Error Navigation to Source	1-7
Error Navigation to Solution Record	1-8

Snapshots	1-8
Creating a Snapshot	1-8
Restoring a Snapshot	1-9
Viewing a Snapshot	1-9
Project Archives	1-9
Creating an Archive	1-9
Restoring an Archive	1-9
Overview of Synthesis Tools	1-10
Xilinx Synthesis Technology (XST)	1-10
Supported Devices	1-10
Process Properties	1-10
FPGA Express	1-10
Supported Devices	1-10
Process Properties	1-11
Synplify/Pro	1-11
Supported Devices	1-11
Leonardo Spectrum	1-11
Supported Devices	1-11
Process Properties	1-12
Design Description	1-12
Design Entry	1-14
Adding Source Files	1-14
Correcting HDL errors	1-15
Starting the HDL Editor	1-16
Creating an HDL-Based Module	1-16
Using the HDL Design Wizard and HDL Editor	1-16
Using the Language Templates	1-18
Creating a CoreGEN Module	1-21
Creating the Core Generator module	1-21
Instantiating the Coregen Module in the HDL Code	1-25
Synthesizing the Design	1-29
Synthesizing the Design using XST	1-30
Synthesizing the Design using FPGA Express	1-32
The Express Constraints Editor (FPGA Express Only)	1-33
Using the Express Constraints Editor (FPGA Express Only)	1-35
Viewing Synthesis Results (FPGA Express Only)	1-38
Synthesizing the Design using Synplify/Synplify Pro	1-40
Synplify's Constraints Editor, SCOPE	1-41
Using Synplify's Constraints Editor, SCOPE	1-42
Examining Synthesis Results	1-45
Synthesizing the Design using Leonardo Spectrum	1-47

Chapter 2 Schematic-Based Design

Getting Started	2-2
Required Software	2-2
Installing the Tutorial	2-2
Tutorial Project Directories and Files	2-3
Copying the Tutorial Files (Optional)	2-3
Starting the ISE Software	2-3
Overview of Project Navigator	2-5
Sources in Project Window	2-5
Module View	2-5
Snapshot View	2-6
Library View	2-7
Processes for Current Source Window	2-7
Process View	2-7
Console Window	2-8
Error Navigation to Source	2-8
Error Navigation to Solution Record	2-8
Snapshots	2-8
Creating a Snapshot	2-8
Restoring a Snapshot	2-8
Viewing a Snapshot	2-9
Overview of Synthesis Tools	2-9
Xilinx Synthesis Technology (XST)	2-9
Supported Devices	2-9
Process Properties	2-9
FPGA Express	2-10
Supported Devices	2-10
Process Properties	2-10
Leonardo Spectrum	2-10
Supported Devices	2-10
Process Properties	2-11
Design Description	2-11
Design Entry	2-14
Starting the Schematic Editor	2-14
Manipulating the Screen View	2-16
Creating a Schematic-Based Macro	2-17
Creating the CNT60 Schematic	2-18
Connectivity—I/O Markers	2-18
Project Libraries	2-19
Adding Components to CNT60	2-19
Correcting Mistakes	2-21

Placing the Remaining Components	2-22
Drawing Wires	2-22
Adding Buses.....	2-23
Adding Bus Taps	2-24
Adding Net Names.....	2-26
Adding I/O Markers.....	2-26
Saving the Schematic	2-27
Creating the CNT60 symbol.....	2-27
Placing the CNT60 Macro.....	2-28
Creating a CORE Generator Module	2-29
Creating the Core Generator module	2-29
Creating a State Machine Module.....	2-32
Opening the State Editor	2-32
Adding New States	2-34
Adding a Transition	2-35
Adding a State Action	2-36
Adding a State Machine Reset Condition	2-37
Creating the State Machine Macro	2-38
Placing the STMACH, Tenths, and decode symbols	2-39
Creating an HDL-Based Module	2-40
Using the HDL Design Wizard and HDL Editor	2-40
Using the Language Templates.....	2-43
Creating the HEX2LED symbol	2-45
Adding the HEX2LED Component to the Schematic.....	2-46
Specifying Device Inputs/Outputs	2-47
Hierarchy Push/Pop.....	2-47
Adding Input Pins.....	2-49
Adding I/O Markers and Net Names	2-50
Assigning Pin Locations.....	2-51
Completing the Schematic.....	2-53

Chapter 3 Behavioral Simulation

Overview of Behavioral Simulation Flow.....	3-1
Required Files.....	3-2
Xilinx Simulation Libraries.....	3-2
Unisims Library	3-2
LogiBLOX Library (VHDL Only)	3-3
XilinxCoreLib Library.....	3-3
Adding an HDL Testbench	3-4
VHDL design.....	3-4
Verilog design	3-4
Creating a Testbench Waveform using HDL Benchner	3-5

Creating a Testbench Waveform Source	3-5
Initializing Inputs	3-6
Generating Expected Results	3-7
Behavioral Simulation using ModelSim	3-8
Selecting Simulation Processes	3-8
Specifying Simulation Properties	3-9
Performing Simulation	3-11
Adding Signals	3-12
Saving the Simulation	3-14
Restarting the Simulation	3-15

Chapter 4 Design Implementation

Installing the Tutorial Files	4-2
Creating an Implementation Project	4-3
Specifying Options	4-4
Translating the Design	4-8
Using the Constraints Editor	4-9
Mapping the Design	4-12
Using the Floorplanner	4-14
Using Timing Analysis to Evaluate Block Delays After Mapping....	4-18
Estimating Timing Goals with 50/50 Rule	4-18
Report Paths in Timing Constraints Option	4-18
Placing and Routing the Design	4-20
Using FPGA Editor to Verify the Place and Route	4-21
Evaluating Post-Layout Timing	4-23
Creating Configuration Data	4-25
Using the PROM File Formatter	4-27

Chapter 5 Timing Simulation

Overview of Timing Simulation Flow	5-1
Required Files	5-2
Xilinx Simulation Libraries	5-2
Starting Modelsim	5-3
Specifying Simulation Process Properties	5-3
Simulation Properties	5-3
Display Properties	5-4
Simulation Model Properties	5-4
Performing Simulation	5-6
Adding Signals	5-6
Saving the Simulation	5-8

Chapter 6 iMPACT Tutorial

Device Support.....	6-2
Download Cable Support	6-2
Parallel Cable III.....	6-2
Multilink Cable.....	6-2
Configuration Mode Support	6-3
Starting the Software.....	6-3
Opening iMPACT from the Project Navigator	6-3
Opening iMPACT stand-alone	6-8
Connecting to a Cable.....	6-10
Boundary Scan Configuration Mode	6-12
Automatically Creating the Chain.....	6-12
Manually Creating the Chain.....	6-14
Assigning Configuration Files	6-14
Saving the Chain Description.....	6-16
Edit Preferences	6-17
Available Boundary Scan Operations	6-17
Performing Boundary Scan Operations	6-19
Troubleshooting Boundary Scan Configuration	6-22
Creating a SVF or STAPL File	6-23
Creating the Chain	6-23
Select Programming File.....	6-23
Writing to the SVF or STAPL File	6-24
Slave Serial Configuration Mode.....	6-26
Adding a Device.....	6-26
Programming the Device	6-30
Troubleshooting Slave Serial Configuration.....	6-32
Select MAP Configuration Mode	6-33
Adding a Device.....	6-34
Programming and Verifying a Device	6-37
Troubleshooting Select MAP Programming and Verify.....	6-39

HDL-Based Design

This chapter guides you through a typical HDL-based design procedure using a design of a runner's stopwatch. The design example used in this tutorial demonstrates many device features, software features and design flow practices you can apply to your own design. This design targets a Virtex-II device; however, all of the principles and flows taught are applicable to any Xilinx device family, unless otherwise noted.

The design is composed of HDL elements and a CORE Generator macro; you can synthesize the design using Xilinx Synthesis Technology (XST), FPGA Express, Leonardo Spectrum, or Synplify.

This chapter is the first in the [“HDL Design Flow.”](#) This chapter is followed by the [“Behavioral Simulation” chapter](#), in which you simulate the HDL code using the ModelSim Simulator. In the [“Design Implementation” chapter](#), you will implement the design using the Xilinx Implementation Tools. The simulation, implementation, and bitstream generation are described in subsequent chapters.

This chapter includes the following sections:

- [“Getting Started”](#)
- [“Overview of Project Navigator”](#)
- [“Overview of Synthesis Tools”](#)
- [“Design Description”](#)
- [“Design Entry”](#)
- [“Synthesizing the Design”](#)

Note: For an example of how to design with CPLDs, go to the online help in Project Navigator. To do so, select **Help** →

ISE Help Contents, and under **Tutorials**, select **CPLD Design Flows**.

Getting Started

The following subsections describe the basic requirements for running the tutorial.

Required Software

The Xilinx Series ISE package is required to perform this tutorial. The design requires that you have installed the Virtex-II libraries and device files, and that you are licensed for FPGA Express or Base Express. You must also have the Watch Tutorial projects which may be downloaded from <http://support.xilinx.com>.

Note: An Express license is required to access the Express Constraints GUI.

Installing the Tutorial

This tutorial assumes that the software is installed in the default location C:\XILINX. If you have installed the software in a different location, substitute your installation path for C:\XILINX.

Unzip the tutorial projects in the C:\XILINX\ directory and replace any existing files. The files downloaded from the web have been updated.

Note: For detailed instructions, refer to the *ISE 4.1i Install and Release Document*.

Tutorial Project Directories and Files

The wtut_vhd and wtut_ver directories are created within C:\XILINX\ISEexamples, and the tutorial files are copied into these directories. These directories contain complete and incomplete versions of the design, done in VHDL and Verilog, respectively. These projects will be used to step through the ISE flow. However, for reference, completed projects are also provided. The following table lists the associated project.

Table 1-1 Tutorial Project Directories

Directory	Description
wtut_vhd	Incomplete Watch Tutorial - VHDL
wtut_ver	Incomplete Watch Tutorial - Verilog
watchvhd_u	Solution for Watch - VHDL (UNIX)
watchver_u	Solution for Watch - Verilog (UNIX)
watchvhd	Solution for Watch - VHDL
watchver	Solution for Watch - Verilog

The watchvhd(_u) and watchver(_u) solution projects contain the design files for the completed tutorials, including HDL files and the bitstream file. To conserve disk space, some intermediate files are not provided. Do not overwrite any files in the solutions directories.

The wtut_vhd and wtut_ver projects contain incomplete copies of the tutorial design. You will create the remaining files when you perform the tutorial. As described in a later step, you have the option to copy the Watch project to another area and perform the tutorial in this new area if desired.

Starting the ISE Software

To follow along with this tutorial, you will need to launch the ISE software package. To do so:

1. Double-click the ISE Project Navigator icon on your desktop or select **Start** → **Programs** → **Xilinx Series ISE 4.1i** → **Project Navigator**.



2. From Project Navigator, select **File** → **Open Project**.

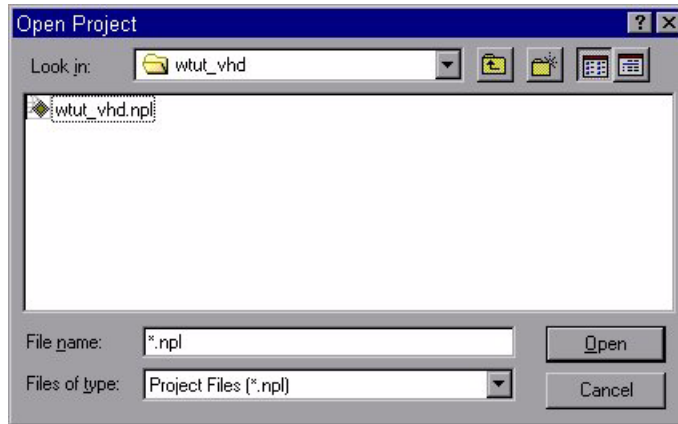


Figure 1-1 Getting Started Dialog Box

3. In the Directories list, browse to c:\xilinx\iseexamples\wtut_vhd or wtut_ver.
4. Double-click on wtut_vhd.npl or wtut_ver.npl.

VHDL or Verilog?

This tutorial has been prepared for both VHDL and Verilog designs. This document applies to both designs simultaneously, noting differences where applicable. You will need to decide which HDL language you would like to work through the tutorial when you open the project.

Overview of Project Navigator

The Project Navigator controls all aspects of the design flow. Through the Project Navigator, you can access all of the various design entry and design implementation tools. You can also access the files and documents associated with your project. The Project Navigator maintains a flat directory structure; therefore, the user must maintain revision control through the use of snapshots.

The Project Navigator is divided into four main subwindows. On the top left is the Sources in Project window which hierarchically displays the elements included in the project. Beneath the Sources in Project window is the Processes for Current Source window which

displays available processes. The third window at the bottom of the Project Navigator is the Message Console and shows status messages, errors, and warnings and is updated during all project actions. The fourth window to the right is the HDL Editor. From this window a user edits source files and accesses the Language Templates. These windows are discussed in more detail in the following sections.

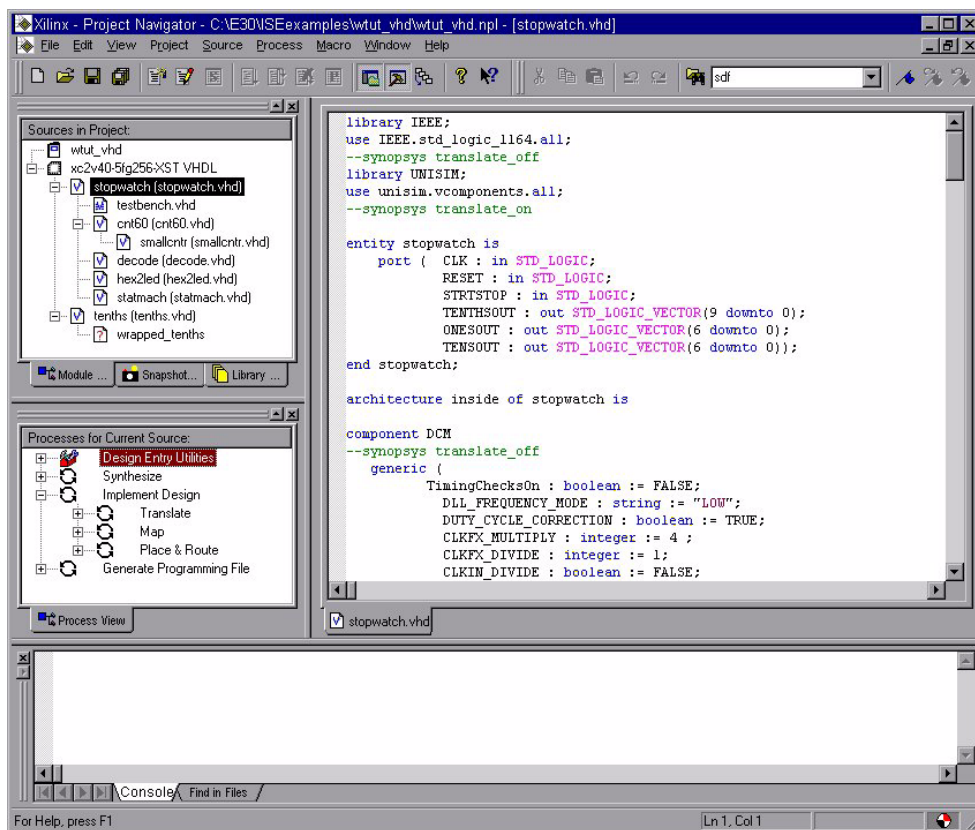


Figure 1-2 Project Navigator

Sources in Project Window

This window has three tabs which provide information for the user. Each tab is discussed in further detail below.

Module View

In the Module View tab of the Sources in Project window, user documents, part type, synthesis tool, and design source files are displayed. User documents are listed under the project name. Source files are listed under the part name and synthesis tool. Next to each filename is an icon which tells you the file type (HDL file, schematic, core, text file, for example). If a file contains lower levels of hierarchy, the icon has a + to the left of the name. HDL files have this + to show the entities (VHDL) or modules (Verilog) within the file. You can expand the tree by clicking this icon. You can open a file for editing by double-clicking on the filename.

Note: While in the module view, you may select a different synthesis tool by double-clicking the project properties (the line above the stopwatch top-level source), and then changing the design flow to another tool.

Snapshot View

A snapshot is a method of revision control. At any time in the design cycle the user can take a snapshot. A snapshot consist of all files in the current working directory. This also includes synthesis and simulation sub-directories. A snapshot can also be restored to resume work at that phase in the design cycle. In the Snapshot View tab of the Sources in Project window, all of the snapshots associated with the open project are displayed. This allows the user to view the reports, user documents, and source files. All information displayed in the snapshot view is read-only.

Note: Remote sources are not copied with the snapshot. A reference is maintained in the snapshot.

Library View

In the Library View tab of the Sources in Project window, all libraries associated with the project are displayed.

Processes for Current Source Window

This window contains the Process View tab.

Process View

The Processes for Current Source Window is located beneath the Sources in Project Window. This window is context sensitive and changes based upon the selected source. The status of each process is displayed on the process icon as a red x, yellow exclamation, or green check mark. The Process Window provides access to the following functions:

- Design Entry Utilities—Provides access to symbol generation, user constraints, and instantiation templates.
- Synthesize—Provides access to check syntax, synthesis, and synthesis reports. This also varies depending on the synthesis tools being used.
- Implement Design— Provides access to implementation tools, design flow reports, and point tools.
- Generate Programming File—Provides access to the configuration tools and bitstream generation.

The Processes for Current Source window incorporates automake technology. This allows the user to select any process in the flow and the software automatically runs the processes necessary to get to the desired step. For example, if the synthesis process has not been run, it is not necessary to run the synthesis process before running the implementation process. Running the implementation process causes the synthesis process to be run before running implementation.

Console Window

Errors, warnings, and informational messages are displayed in the Console Window. Errors and warnings are signified by a red box next to the message, while warnings have a yellow box.

Error Navigation to Source

The Console Window provides the ability to navigate from a synthesis error or warning message to the source HDL file. This can be done by selecting the error or warning message, right-clicking the

mouse and selecting Goto Source. This will open the HDL source file and move the cursor to the line with the error.

Error Navigation to Solution Record

The Console Window provides the ability to navigate from an error or warning message to the support.xilinx.com web site. These type of errors or warnings can be identified by the web icon to the left of the error. To navigate to the solution record, select the error or warning message, right-click the mouse and select Goto Solution Record. This opens a web browser and displays all solution records applicable to this message.

Snapshots

Snapshots provide the user the ability to maintain revision control over the design. A snapshot contains all of the files in the project directory.

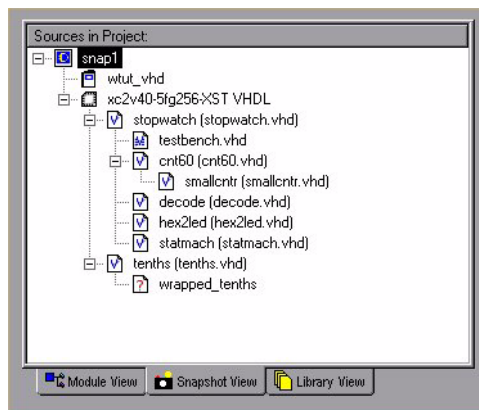


Figure 1-3 Snapshot View

Creating a Snapshot

A snapshot is created by selecting **Project** → **Take a Snapshot**. This opens the Take a Snapshot of the Project dialog box. This allows the user to enter the snapshot name and any comments associated with the snapshot. The snapshot contains all of the files in the project directory along with project settings.

Restoring a Snapshot

The Snapshot View, of the Source Window, contains a list of all the snapshots available in the current project. Since snapshots are read-only, a snapshot must be restored in order to continue work. To do this, select the snapshot and select **Project** → **Replace with Snapshot**. The user is prompted to create a snapshot of the current project directory and restore the selected snapshot for further work.

Viewing a Snapshot

The Snapshot View, of the Source Window, contains a list of all the snapshots available in the current project. A snapshot can be opened to review report or verify process status by selecting the snapshot, right-click the mouse and selecting Open.

Project Archives

The ISE software also allows a user to archive the entire project into a single compressed file. This allows for easier transfer over email and storage of numerous projects in a limited space.

Creating an Archive

An archive can be created by selecting **Project** → **Archive**. This opens the Create Zip Archive dialog box, in which the user enters the archive name and location to be saved. The archive contains all of the files in the project directory along with project settings. Remote sources are not zipped up into the archive.

Restoring an Archive

ISE does not have a specific menu item or feature to restore an archive, as the compressed file can be extracted with any ZIP utility. The project directory will be placed in the location where the archive is extracted.

Overview of Synthesis Tools

This tutorial explains and demonstrates how to synthesize your design using four synthesis tools. The following section lists the devices supported by each synthesis tool and includes some process properties information.

Xilinx Synthesis Technology (XST)

Supported Devices

- Virtex™/-E /-II /-IIPro
- Spartan™-II /-IIE
- XC9500™ /XL/XV
- Coolrunner™ /-II

Process Properties

Process properties allow the user to control the synthesis results of XST. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties the user can control the synthesis results for area or speed, and the amount of time the synthesizer runs.

More detailed information is available in the XST User Guide.

FPGA Express

Supported Devices

- Virtex™/-E /-II /-IIPro
- Spartan™/XL/-II/-IIE
- XC9500™ /XL /XV
- XC4000™ E /EX /XL /XV /XLA
- Coolrunner™ /-II

Process Properties

Process properties allow you to control the synthesis results of FPGA Express. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties, you can control the synthesis results for area or speed and the amount of time the synthesizer runs.

More detailed information is available in the FPGA Express online help.

Synplify/Pro

This synthesis tool is not part of the ISE package and is not available unless purchased separately.

Supported Devices

- Virtex™/-E /-II /-IIPro
- Spartan™/XL/-II/-IIE
- XC9500™ /XL /XV
- XC4000™E /EX /XL /XV /XLA
- Coolrunner™ /-II

Leonardo Spectrum

This synthesis tool is not part of the ISE package and is not available unless purchased separately.

Supported Devices

- Virtex™/-E /-II /-IIPro
- Spartan™/XL/-II/-IIE
- XC9500™ /XL /XV
- XC4000™E /EX /XL /XV /XLA
- Coolrunner™ /-II

Process Properties

Process properties allow you to control the synthesis results of Leonardo Spectrum. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties the user can control the synthesis results for area or speed and the amount of time the synthesizer runs.

More detailed information is available in the Leonardo Spectrum online help.

Design Description

The design used in this tutorial is a hierarchical, HDL-based design which means that the top-level design file is an HDL file that references several other lower-level macros. The lower-level macros are either HDL modules or CORE Generator modules.

The design begins as an unfinished design. Throughout the tutorial, you complete the design by generating some of the modules from scratch and by completing some others from existing files. When the design is complete, you simulate it to verify the design's functionality.

The Watch design is a simple runner's stopwatch. There are three external inputs, and three external output buses in the completed design. The system clock is an externally generated signal. The following list summarizes the input lines and output buses.

Inputs

- **STRTSTOP**—Starts and stops the stopwatch. This is an active low signal which acts like the start/stop button on a runner's stopwatch.
- **RESET**—Resets the stopwatch to 00.0 after it has been stopped.
- **CLK**—Externally generated system clock.

Outputs

- TENSOUT[6:0]—7-bit bus which represents the Ten's digit of the stopwatch value. This bus is in 7-segment display format viewable on the 7-segment LED display.
- ONESOUT[6:0]—Similar to TENSOUT bus above, but represents the One's digit of the stopwatch value.
- TENTHSOUT[9:0]—10-bit bus which represents the Tenths' digit of the stopwatch value. This bus is one-hot encoded.

The completed design consists of the following functional blocks.

- STATMACH
State Machine module.
- CNT60
HDL-based module which counts from 0 to 59, decimal. This macro has 2 4-bit outputs, which represent the ones and tens digits of the decimal values, respectively.
- TENTHS
CORE Generator 4-bit, binary encoded counter. This macro outputs a 4-bit code which is decoded to represent the tenths digit of the watch value as a 10-bit one-hot encoded value.
- HEX2LED
HDL-based macro. This macro decodes the ones and tens digit values from hexadecimal to 7-segment display format.
- SMALLCNTR
A simple Counter.
- DECODE
Decoded the CORE Generator output from 4-bit binary to a 10-bit one hot output.

Design Entry

In this hierarchical design, you will examine HDL files, correct syntax errors, create an HDL macro, and add a CORE Generator module. In this tutorial, you will create and use each type of design macro so that you can apply these procedures to your own design.

With `wtut_vhd` or `wtut_ver` project open in Project Navigator, the Sources in Project window displays all of the source files currently added to the project, with the associated entity or module names (see [Figure 1-4](#)). In the current project, `smallcntr` and `hex2led` are instantiated, but the associated entity or module is not defined in the project. Instantiated components with no entity or module declaration are displayed with a red question-mark.

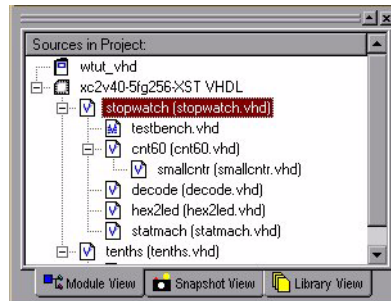


Figure 1-4 Sources in Project Window

Adding Source Files

HDL files must be added to the project before they can be synthesized. Four HDL files have already been added to this project, but have not yet been analyzed. To analyze the source files,

1. Select `stopwatch.vhd` or `stopwatch.v` in the Sources in Project window.

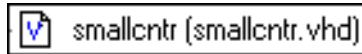
Upon selecting the HDL file, the process window displays all processes available for this file.

Now add the remaining HDL file to the project.

2. Select **Project** → **Add Source**.
3. Select `smallcntr.vhd` or `smallcntr.v` from the project directory.

4. In the Choose Source Type dialog box, select HDL module.
5. Click **OK**.

The red question-mark (?) for smallcntr should change to a **V**.



After adding the file to the project, the file is not automatically analyzed. To analyze the source files:

1. Select stopwatch.vhd or stopwatch.v in the Sources in Project window.

Upon selecting the HDL file, the Processes for Current Sources in Project window displays all processes available for this file.

2. Select Analyze Hierarchy in the Synthesize hierarchy to update the files.

Correcting HDL errors

The SMALLCNTR design contains a syntax error that must be corrected. The red “x” beside the Analyze Hierarchy process indicates an error was found during analysis. The Project Navigator reports errors in red and warnings in yellow in the console.

Note: Help for FPGA Express errors or warning is available in the stand alone version of Express.

To display the error in the source file:

3. Double-click on the error message in the console window.
4. Correct any errors in the HDL source file. The comments next to the error explain this simple fix.
5. Select **File** → **Save** to save the file.
6. Re-analyze the file by selecting the HDL file and double-clicking Analyze Hierarchy under the Synthesize hierarchy to update these file.

Starting the HDL Editor

There are three different ways to open the HDL Editor tool.

- **File** → **New** opens an untitled file in the HDL Editor.
- Double-click an HDL file in the Sources in Project window from the Module View, File View, Snapshot View, or Library View.
- Right-click an HDL file in the Sources in Project window and select Open.

You may stop the tutorial at any time and save your work by selecting **File** → **Save All**.

Creating an HDL-Based Module

With ISE, you can easily create modules from HDL code. The HDL code is connected to your top-level HDL design through instantiation and compiled with the rest of the design.

You will create a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 7-segment LED display format.

Using the HDL Design Wizard and HDL Editor

You enter the name and ports of the component in HDL Wizard, and HDL Wizard creates a “skeleton” HDL file which you can complete with the remainder of your code.

To create the source file:

1. Select **Project** → **New Source**.

A dialog box opens in which you specify the type of source you want to create.

2. Select VHDL or Verilog Module.
3. In the File Name field, type ‘hex2led’.
4. Click on **Next**.

The hex2led component has a 4-bit input port named hex and a 7-bit output port named led. To enter these ports:

1. Click in the Port Name field and type HEX.
2. Click in the Direction field and set the direction to in.
3. In the MSB field enter 3, and in the LSB field enter 0.

Port Name	Direction	MSB	LSB
HEX	in	3	0
LED	out	6	0
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Figure 1-5 HDL Wizard

Repeat the previous steps for the LED[6:0] output bus. Be sure that the direction is set to out.

4. Click **Next** to complete the Wizard session.

A description of the module is now displayed.

5. Click **Finish** to open the “skeleton” HDL file in HDL Editor.

The skeleton VHDL and Verilog HDL file are found in [Figure 1-6](#) and [Figure 1-7](#).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hex2led is
    Port ( LED : in std_logic_vector(3 downto 0);
          HEX : out std_logic_vector(6 downto 0));
end hex2led;

architecture behavioral of hex2led is

begin

end behavioral;
```

Figure 1-6 Skeleton VHDL File

```
module hex2led ( HEX,LED);
    input [3:0] HEX;
    output [6:0] LED;

endmodule
```

Figure 1-7 Skeleton Verilog File

In the HDL Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are printed in blue, data types in red, comments in green, and values are black. This color-coding enhances readability and recognition of typographical errors.

Using the Language Templates

The ISE language templates include HDL constructs and synthesis templates which represent commonly used logic components, such as counters, D flip-flops, multiplexers, and primitives. The instantiation templates created by the CORE Generator are placed among the language templates in a COREGEN folder. You can add your own

templates to the language template for components or constructs you use often.

To invoke the Language Assistant and select the template for this tutorial:

1. Select **Edit → Language Templates**.

Each HDL language in the Language Template is divided into four sections: Component Instantiations, Language Templates, Synthesis Templates, and User Templates. To expand the view of any of these sections, click the + next to the topic. Click any of the listed templates to view the template in the right-hand pane.

2. Under either the VHDL or Verilog hierarchy, expand the Synthesis Templates hierarchy and select the template called HEX2LED Converter. Use the appropriate template for the language you are using.
3. To preview the HEX2LED Converter template, click the template in the hierarchy and the contents are displayed in the right-hand pane.

This template provides source code to convert a 4-bit value to 7-segment LED display format.

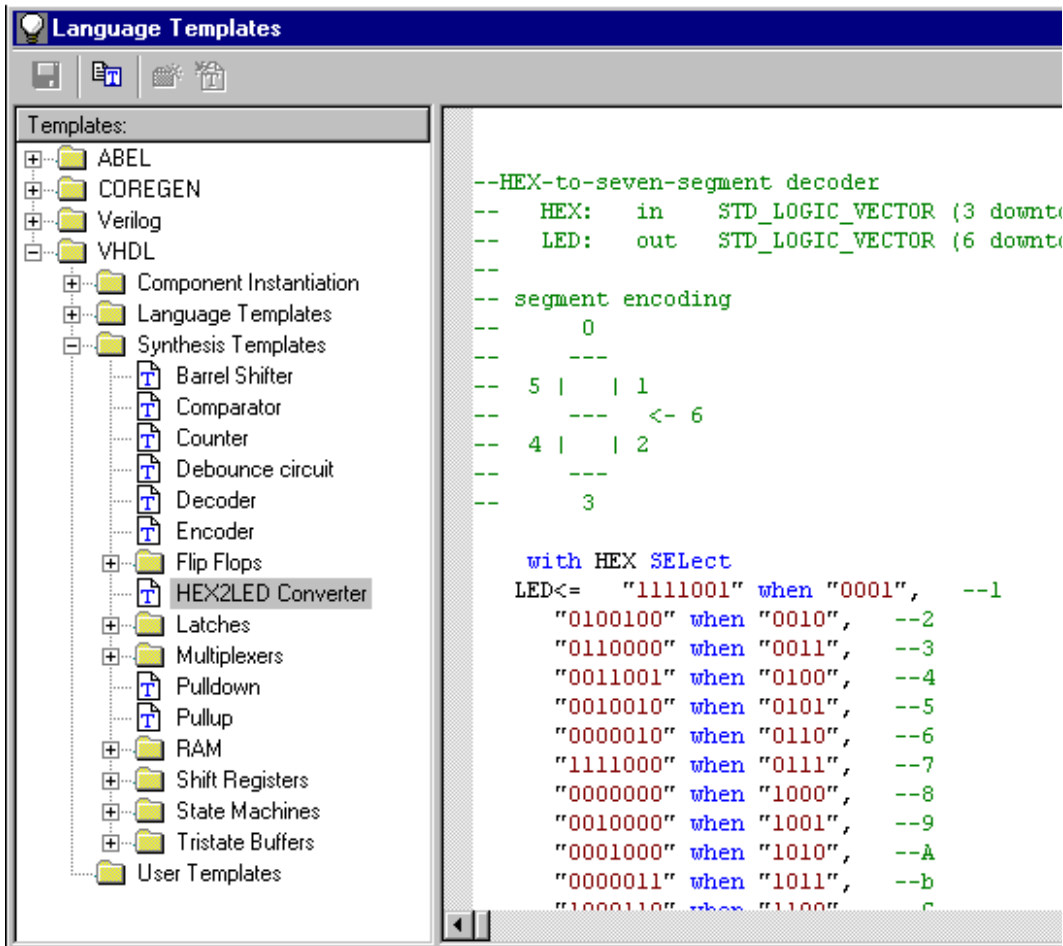


Figure 1-8 Language Templates

The tutorial describes the drag and drop method only for adding templates to your HDL file. Please note that you can also copy and paste directly from the Language Template and use the right-click menu.

To add the template to your HDL file:

1. In the Language Template, click and drag the HEX2LED Converter name into the hex2led.vhd file under the architecture statement, or the hex2led.v file under the module declaration.
2. Close the Language Assistant by clicking the X in the upper-right corner of the window.
3. (Verilog only) After the input and output statements and before the HEX2LED converter that you just added, add the following line of code to the HDL file to allow an assignment:

```
reg LED;
```

You now have complete and functional HDL code.

4. Save the file by selecting **File** → **Save**.
5. Select hex2led in the Sources in Project window and double-click Check Syntax under Synthesize in the Processes for Current Source window.
6. Exit the HDL Editor.

Creating a CoreGEN Module

CORE Generator is a graphical interactive design tool you use to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and pre-optimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic for arithmetic functions, and on-chip RAM for dual-port and synchronous RAM.

In this section, you create a CORE Generator module called TenthS. TenthS is a 4-bit binary encoded counter. The 4-bit number is decoded to count the tenths digit of the stopwatch's time value.

Creating the Core Generator module

Select the type of module you want in the CORE dialog box, as well as the specific features of the module. Invoke this GUI from the Project Navigator New Source Wizard.

To select the module type:

1. In Project Navigator, select **Project** → **New Source**.
2. Select Coregen IP.
3. Enter tenths in the File Name field.
4. Click **Next** and then **Finish**.

The Xilinx CORE Generator 4.1i opens and displays a list of possible COREs available.

5. Double-click on Basic Elements - Counters.
6. Double-click on Binary Counter to open the Binary Counter dialog box.

This dialog box allows you to customize the counter to the design specifications.

7. Fill in the Binary Counter dialog with the following settings:
 - Component Name: tenths
Defines the name of the module.
 - Output Width: 4
Defines the width of the output bus.
 - Operation: Up
Defines how the counter will operate. This field is dependant on the type of module you select.
 - Count Style: Count by Constant
Allows counting by a constant or a user supplied variable.
 - Count Restrictions: Enable and Count To Value A (HEX)
This dictates the maximum count value.
 - Threshold Options: Threshold0 set to A
Signal goes high when the value specified has been reached.
 - Threshold Options: Registered
8. Click the **Register Options** button to open the Register Options dialog box.
9. In the Register Options dialog box, enter the following settings:

- Clock Enable: Selected
- Asynchronous Settings: Init with a value of 1.
- Synchronous Settings: None

10. Click **OK**.

Note: Check that *only* the following pins are used:

- AINIT
- CE
- Q
- Q_Thresh0
- CLK

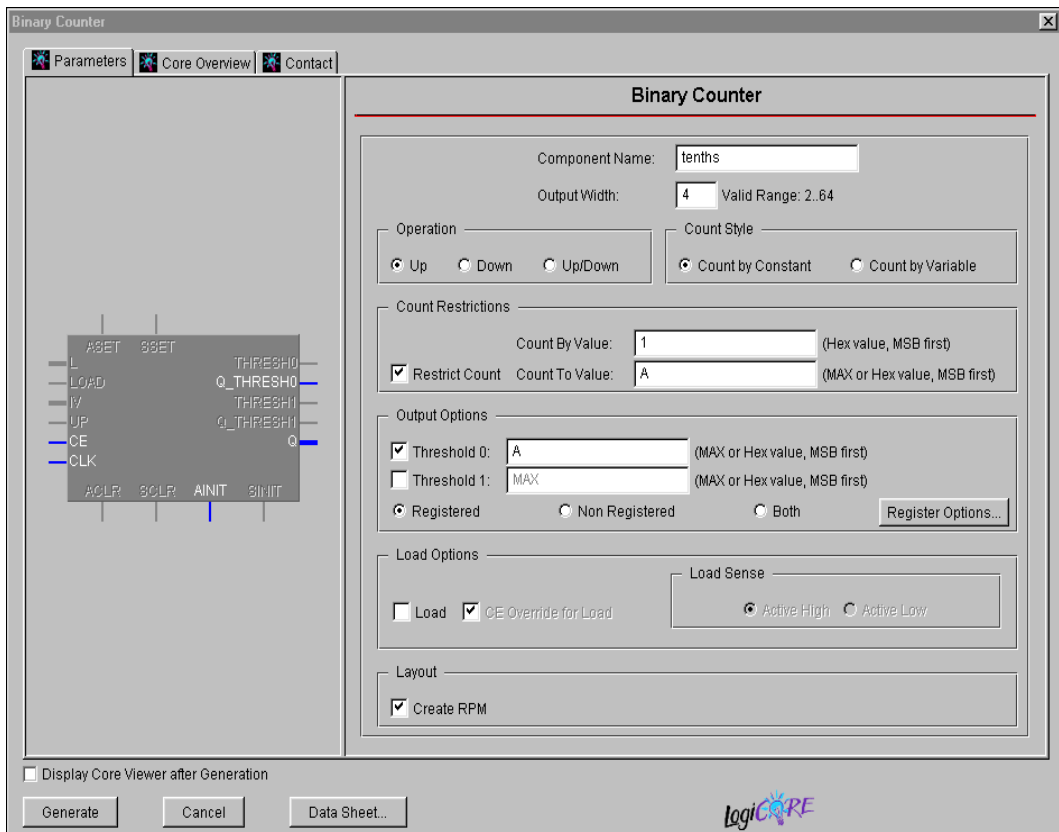


Figure 1-9 CoreGen Module Selector

11. Click **Generate**.

The module is created and automatically added to the project library.

A number of files are added to the project directory. These files are:

- tenths.edn

This file is the netlist that is used during the Translate phase of implementation.

- tenths.vhd or tenths.v

This is the instantiation template that is used to incorporate the CORE Generator module in your source HDL.

- tenths.xco

This file stores the configuration information for the Tenths module and is used as a project source.

- coregen.prj

This file stores the Coregen configuration for the project.

12. Click **Cancel** and close Core Generator.

Instantiating the Coregen Module in the HDL Code

Next, instantiate the Coregen Module in the HDL code using either a VHDL flow or a Verilog flow.

VHDL Flow

To instantiate the Coregen Module using a VHDL flow:

1. Double-click stopwatch.vhd to open the file in HDL Editor.
2. Place your cursor after the line that states:

“-- Insert Coregen Counter Component Declaration”

3. Select **Edit** → **Insert File** and choose Tenths.vhd.

The VHDL template file for the Coregen instantiation is inserted.

Note: The Component Declaration does not need to be modified.

4. Highlight the inserted code from
“-- Begin Cut here for INSTANTIATION Template”
to
“AINIT=>AINIT);”
5. Select **Edit** → **Cut**.

```
architecture inside of stopwatch is

component statmach
    port ( CLK : in STD_LOGIC;
          RESET : in STD_LOGIC;
          STRTSTOP : in STD_LOGIC;
          CLKEN : out STD_LOGIC;
          RST : out STD_LOGIC);
end component;

-- Insert Coregen Counter Component Declaration.

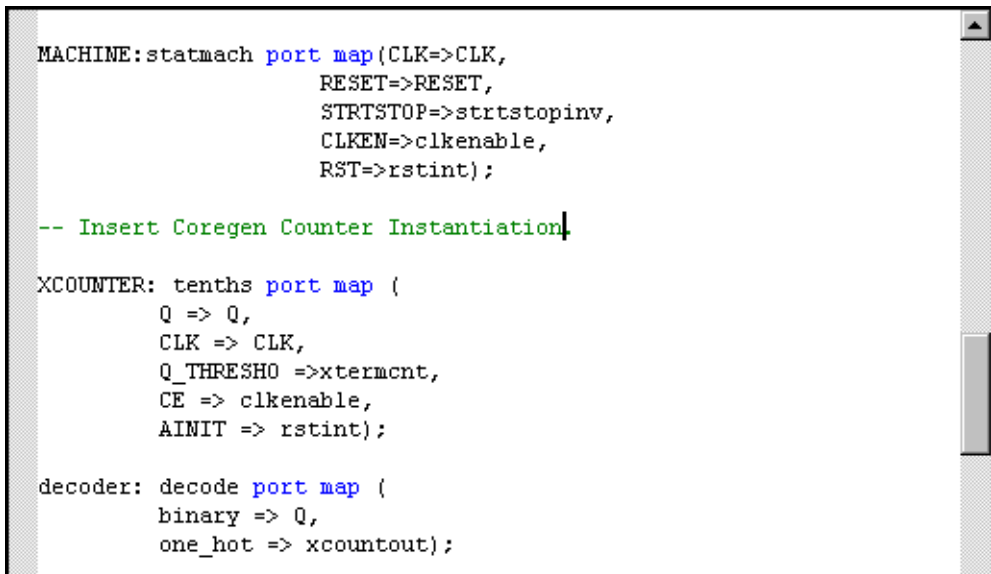
component tenths
    port (
        Q: OUT std_logic_vector(3 downto 0);
        CLK: IN std_logic;
        Q_THRESHO: OUT std_logic;
        CE: IN std_logic;
        AINIT: IN std_logic);
end component;

component decode port (
    binary: in std_logic_vector(3 downto 0);
    one_hot: out std_logic_vector(9 downto 0));
end component;
```

Figure 1-10 VHDL Component Declaration of Coregen Module

6. Place the cursor after the line that states:
“--Insert Coregen Counter Instantiation”
7. Select **Edit** → **Paste** to place the instantiation here.
8. Change “your_instance_name” to XCOUNTER.
9. Edit this instantiated code to connect the signals in the Stopwatch design to the ports of the Coregen module.

The completed code looks like the following:



```

MACHINE: statmach port map (CLK=>CLK,
                           RESET=>RESET,
                           STRTSTOP=>strtstopinv,
                           CLKEN=>clkenable,
                           RST=>rstint);

-- Insert Coregen Counter Instantiation

XCOUNTER: tenths port map (
    Q => Q,
    CLK => CLK,
    Q_THRESHO => xtermcnt,
    CE => clkenable,
    AINIT => rstint);

decoder: decode port map (
    binary => Q,
    one_hot => xcountout);
  
```

Figure 1-11 VHDL Component Instantiation of Coregen Module

10. Remove the attributes for the two synthesis tools not used and any comments not relevant to the design.
11. Save the design (**File** → **Save**) and close the HDL Editor.

Verilog Flow

To instantiate the Coregen Module using a Verilog flow:

1. Double-click stopwatch.v to open the file in HDL Editor.
2. Place your cursor after the line that states:
 “// Place the CoreGen Module Declaration for Tenths here”
3. Select **Edit** → **Insert File** and choose Tenths.v.

The Verilog template file for the Coregen instantiation is inserted.

Note: The Component Declaration does not need to be modified.

4. Highlight the inserted code from “Tenths YourInstanceName” to “AINIT=(AINIT));”
5. Select **Edit** → **Cut**.

```
// synopsys translate_off

////////////////////////////////////
//// You must set this path to match your install //////////////////////////////////
`include "D:/xilinx/verilog/src/XilinxCoreLib/C_COUNTER_BINARY_V1_0.v"

// synopsys translate_on

//Place the CoreGen Module Declaration for Tenth's here
module tenths (
    Q,
    CLK,
    Q_THRESH0,
    CE,
    AINIT);

output [3:0] Q;
input CLK;
output Q_THRESH0;
input CE;
```

Figure 1-12 Verilog Module Declaration of Coregen Module

6. Place the cursor after the line that states:
“// Place the CoreGen Component Instantiation for Tenth's here.”
7. Select **Edit** → **Paste** to place the instantiation here.
8. Change “YourInstanceName” to XCOUNTER.
9. Edit this code to connect the signals in the Stopwatch design to the ports of the CoreGen module.

The completed code is shown in the following figure:

```
statmach MACHINE(.clk(CLK),
    .reset(RESET),
    .strtstop(strtstopinv),
    .clken(clkenable),
    .rst(rstint));

//Place the CoreGen Component Instantiation for Tenths here
tenths XCOUNTER (
    .Q(Q),
    .CLK(CLK),
    .Q_THRESHO(xtermcnt),
    .CE(clkenable),
    .AINIT(rstint));
//----- End INSTANTIATION Template -----

decode one_decode (.BINARY(Q), .ONE_HOT(xcountout));
```

Figure 1-13 Verilog Component Instantiation of the CoreGen Module

10. Save the design (**File** → **Save**) and close stopwatch.v in the HDL Editor.

Synthesizing the Design

Up to this point, the design has been utilizing XST for syntax checking and analysis. The synthesis tool can be changed at anytime during the design flow. To change the synthesis tool:

1. Select the targeted part in the Sources in Project window.
2. Select **Source** → **Properties**.
3. In the Project Properties dialog box, click the Synthesis Tool column and use the pulldown arrow to select the desired synthesis tool from the list.

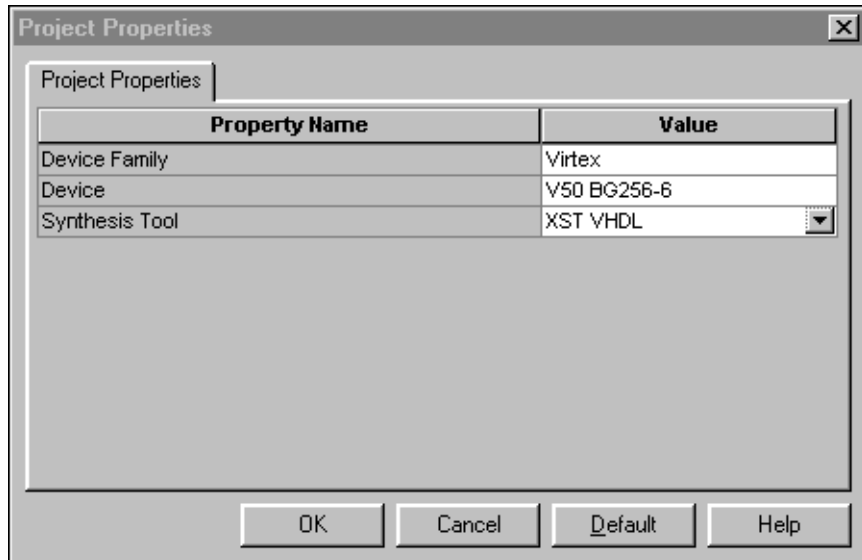


Figure 1-14 Specifying Synthesis Tool

This tutorial describes design synthesis using the tools XST, FPGA Express, Synplify with SCOPE, and Leonardo Spectrum in the following sections:

- [“Synthesizing the Design using XST”](#)
- [“Synthesizing the Design using FPGA Express”](#)
- [“Synthesizing the Design using Synplify/Synplify Pro”](#)
- [“Synthesizing the Design using Leonardo Spectrum”](#)

Synthesizing the Design using XST

Now that the design has been entered and analyzed, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

To synthesize the design with XST:

1. Select stopwatch.vhd (or stopwatch.v).
2. Double-click on the Synthesize process in the Processes for Current Source window.

Note: This step can also be done by selecting stopwatch.vhd (or stopwatch.v), clicking Synthesize in the Processes for Current Source window and selecting **Process** → **Run**.

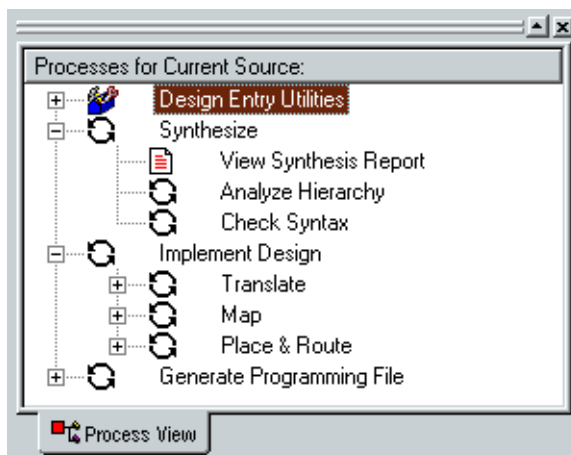


Figure 1-15 Synthesis/Implementation Window

At this point, an EDN file exists for the Stopwatch design. Go to:

- the “[Behavioral Simulation](#)” chapter to perform a pre-synthesis simulation of this design.
- the “[Design Implementation](#)” chapter to place and route the design.
- the “[Timing Simulation](#)” chapter for post-place and route simulation.

Note: For more information concerning XST constraints, options, reports, or running XST via command line see the *XST User Guide* at <http://support.xilinx.com>.

Synthesizing the Design using FPGA Express

Now that the design has been entered and analyzed, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

To synthesize the design with FPGA Express:

1. Set the global synthesis options by selecting stopwatch.vhd (or stopwatch.v), then right-click on the Synthesis process and select Properties.
2. Set the Default Frequency to 50MHz, and check the Export Timing Constraints box.
3. Click **OK** to accept these values.
4. Select stopwatch.vhd (or stopwatch.v) and double-click on the Synthesize process in the Processes for Current Source window.

Note: This step can also be done by selecting stopwatch.vhd (or stopwatch.v), clicking Synthesize in the Processes for Current Source window and selecting **Process** → **Run**.

The process labeled Edit Constraints creates a functional structure of the design and opens the Express Constraints Editor. The process labeled View Synthesis Results optimizes and synthesizes the functional structure, then displays the results in the Express Constraints Editor.

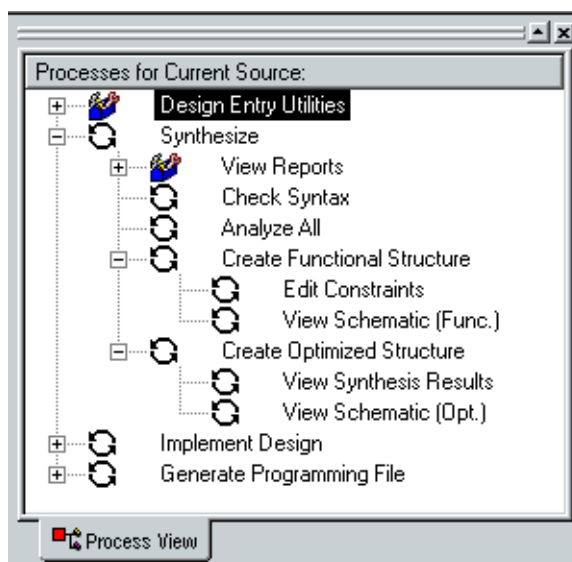


Figure 1-16 Synthesis/Implementation Window

Note: The Express Constraints Editor is not available to users who are not registered or those with a Base Express license. All the functionality covered by the Express Constraints Editor can be achieved by component instantiation (Pullups, Pulldowns, Clock Buffers, I/O Flip-Flops), UCF file (timing constraints, pin location constraints), MAP options (merging flip-flops into IOBs), or through `fe_shell` scripting. If you are a Base Express customer, skip to the [“Behavioral Simulation” chapter](#).

The Express Constraints Editor (FPGA Express Only)

The Express Constraints Editor allows you to control optimization options and pass timing specifications to the Place and Route software. This Editor is only available with the FPGA Express product, and not Base Express. All timing specifications are passed in the netlist directly to the place and route engine, and are used in the synthesis process for timing estimation purposes only.

Clocks

The Default Frequency, set in Synthesize Process Properties, is applied to all clocks in the design.

To change the specification of a clock:

1. Click inside the box to the right of the clock and select Define.
2. Enter the clock period or give the rise and fall times.

Paths

All types of paths that can be covered by timing specifications are listed here, with unique specifications given for each clock in the design.

To modify these specifications, enter a new delay in the Req. Delay column.

To create a subpath within a path:

1. Right-click the source(from) or destination(to) and select New Subpath.
2. Enter the subpath name and delay value.
3. Select sources and destinations by double-clicking the instances. You can also use wildcards in the selection filters to choose a group of elements.

Ports

In the Ports tab, you can set input and out delay requirements, assign clock buffers, insert pullup or pulldown resistors in the I/O, set delay properties for input registers, set slew rate, disable the use of I/O registers, and assign pin locations. For all but the pin locations, click in the box to use the pulldown menu. For pin locations, type the pin number in the box.

Modules

With the Modules tab, you can keep or eliminate hierarchy and disable resource sharing. You can also override the default settings for effort and area versus speed at the module level. Block level Incremental Synthesis is also enabled here.

Registers

This tab allows the designer to view the estimated fanout of the registers as well as setting the maximum fanout for the registers.

Xilinx Options

The Ignore unlinked cells during GSR mapping option directs Express to infer a global reset signal (and, therefore, insert the STARTUP module), even if black boxes have been instantiated. Express does not know the reset characteristics of any logic in black boxes, so it will not insert STARTUP unless you check this option.

Using the Express Constraints Editor (FPGA Express Only)

Xilinx recommends that you let the automatic placement and routing program, PAR, define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place-and-route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a PCB (printed circuit board).

Define the initial pinout by running the place-and-route tools without pin assignments, then lock down the pin placement so that it reflects the locations chosen by the tools. Assign locations to the pins in the Watch design so that the design can function in a Xilinx demonstration board. Because the design is simple and timing is not critical, these pin assignments do not adversely affect the ability of PAR to place-and-route the design.

For HDL-based designs, these pin assignments can be done in a User Constraints File (.ucf) or with the Express Constraints Editor. Although UCF files are provided for this tutorial, you will assign some basic timing constraints with a pin location constraint. You can save the Express Constraints Editor constraints at any time by clicking the Export Constraints icon. Doing so will give you the option of saving the file as an ASCII .ctl file or binary .exc file. If you do not export the constraints, they remain for the particular design version that you are working on.

1. Under the Paths tab, click in the box in Row 1 below the Req. Delay header (from All Input Ports to RC-CLK).
2. Change the delay to 15.

Under the Ports tab, the Input Delays for RESET and STRTSTOP have changed to 15, as these represent all the Pad to Setup delays.

Note: To change the values of individual Input or Output Delays, click the value in the Ports tab and either editing the value there or using the pulldown tab to select a value or define a new one. Change the values on one of the output signals using one of these methods.

3. Under the Paths tab, right-click either RC-CLK or All Output Ports in the second row and select New Sub path.

The Create/Edit Timing Subpath window opens.

4. Give this new subpath a name, Sub_flops_to_out, and a Delay value, 18.
5. On the left-hand side, double-click all four flip-flops that contain the name *sixty/lsbcount/QOUT*, to determine the sources of this subpath.
6. On the lower right hand side, use the filter to select the destinations. Type ONE* in the field and click the **Select** button.

All the ports beginning with ONESOUT will be highlighted.

7. Click **OK** to see your new subpath.

Note: The filter is case sensitive.

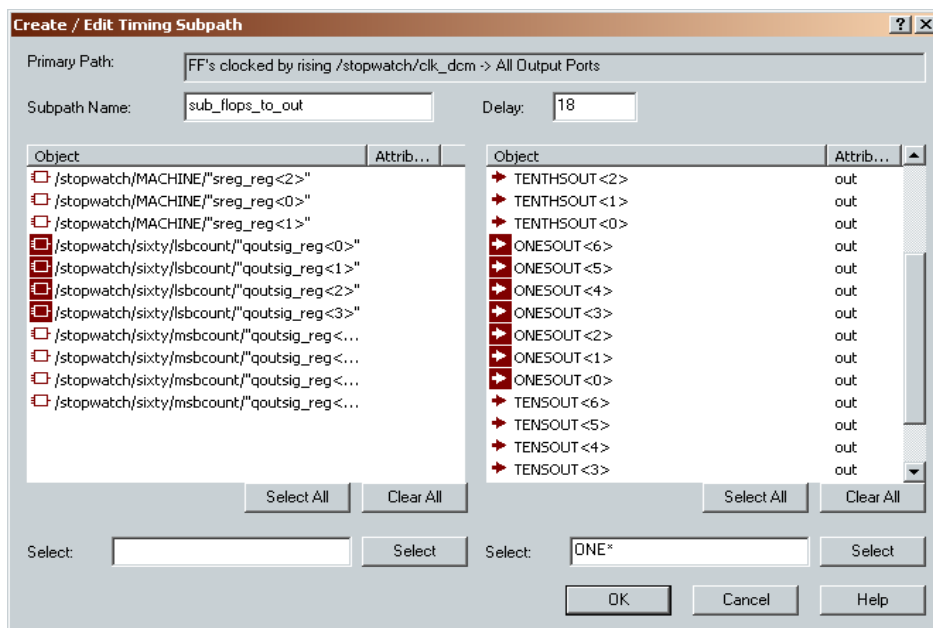


Figure 1-17 Editing Subpath in the Express Constraints Editor

8. Under the Ports tab, add one pin location constraint in the Pad Loc column.
9. Scroll to the right to see this column.

CLK must be assigned to P15. To reassign, click the box and enter the pin number.

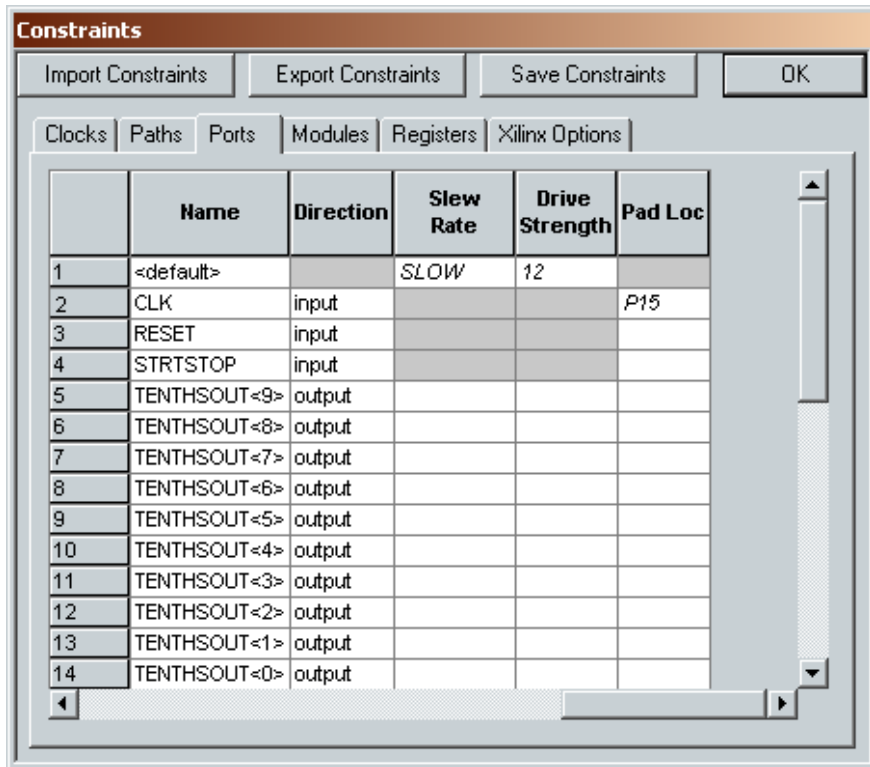


Figure 1-18 Ports Tab Display

10. Click **Save** → **Constraints**.
11. Click **OK** to close the editor.

Viewing Synthesis Results (FPGA Express Only)

Constraint requirements and results can be viewed by double-clicking View Synthesis Results under Synthesize in the Processes for Current Source window. The delay values are based on wireload models and, therefore, must be considered preliminary. Consult the post-route timing reports for the most accurate delay information.

To view the synthesis results:

1. Under the Clocks tab, examine the estimated delay value of the clock. Delays greater than the specification appear in red.
2. Under the Paths tab, examine the estimated delays for the paths and subpath.
3. Click the source or destination of a path to see the members of the path, and click a specific path to see the individual segments of that path.

	From	To	Instance Path	Cell Type	Delay
1	All Input Ports	RC - clk_dcm	qoutsig_reg<0>/C	FDCE	0.0
2	RC - clk_dcm	All Output Ports	qoutsig_reg<0>/Q	FDCE	0.4
3	Sub_flops_to_out-From	Sub_flops_to_out-To	C6/M0	LUT4	0.4
4	RC - clk_dcm	RC - clk_dcm	C6/O	LUT4	0.7
5			C_ONESOUT<6>/I	OBUF_LVTTL_S_12	0.7
6			C_ONESOUT<6>/O	OBUF_LVTTL_S_12	3.5
7			ONESOUT<6>/ONESOUT<6>	OUT	3.5

	Sub_flops_to_out-F rom	Sub_flops_to_out-T o	Est. Del	Slack
1	qoutsig_reg<0>	ONESOUT<6>	3.5 / 1.4	
2	qoutsig_reg<0>	ONESOUT<5>	3.5 / 1.4	
3	qoutsig_reg<0>	ONESOUT<4>	3.5 / 1.4	
4	qoutsig_reg<0>	ONESOUT<3>	3.5 / 1.4	
5	qoutsig_reg<0>	ONESOUT<2>	3.5 / 1.4	
6	qoutsig_reg<0>	ONESOUT<1>	3.5 / 1.4	
7	qoutsig_reg<0>	ONESOUT<0>	3.5 / 1.4	

Figure 1-19 Estimated Timing Data Under Paths Tab

4. Examine the Ports tab to see that all of the settings and delays have been assigned and met.
5. Under the Modules tab, examine the elements used to synthesize this design. Click the box in the second row under Area and click **Details**. This section summarizes all the design elements used in the Stopwatch design that Express knows about.

Since the Tenths module is a CORE Generator component and has not been synthesized by Express, it is UNLINKED and no summary information is available.

Note: Black boxes (modules not read into the Express design environment) are always noted as UNLINKED in the Express reports. As long as the underlying netlist (.xnf, .ngo, .ngc or .edn) for a black box exists in the project directory, the Implementation tools merge the netlist in during the Translate phase. Since the Tenths module was built using Coregen called from the project, the tenths EDN file will be found.

6. Click **OK** to close the editor.

At this point, an EDN file exists for the Stopwatch design. Go to:

- the [“Behavioral Simulation” chapter](#) to perform a pre-synthesis simulation of this design.
- the [“Design Implementation” chapter](#) to place and route the design.
- the [“Timing Simulation” chapter](#) for post place and route simulation.

Synthesizing the Design using Synplify/Synplify Pro

Now that the design has been entered and analyzed, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

To synthesis the design:

1. Set the global synthesis options by selecting stopwatch.vhd (or stopwatch.v), then right-clicking Synthesis in the Processes for Current Source window, and selecting Properties.
2. Set the Default Frequency to 50MHz, and check the Write Vendor Constraint File box.
3. Click **OK** to accept these values.
4. Select stopwatch.vhd (or stopwatch.v) and double-click on the Synthesize process in the Process Window.

Note: This step can also be done by selecting stopwatch.vhd (or stopwatch.v), clicking Synthesize in the Processes for Current Source window and selecting **Process** → **Run**.

Synplify's features include:

- The process labeled Compile will synthesize the RTL into generic logic.
- The process labeled Mapping will map the synthesized RTL into the target technology.
- Synplify's other features such as the RTL viewer and the constraints editor called SCOPE are available from the synthesis process window.

Double-click any one of these features to launch Synplify and open the appropriate feature window.

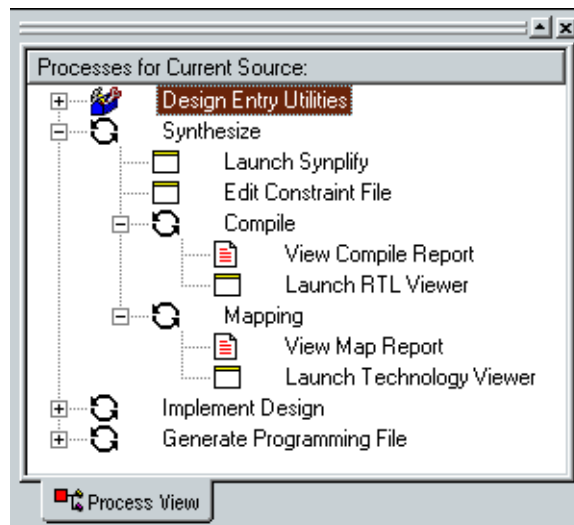


Figure 1-20 Synplify Synthesis/Implementation Window

Synplify's Constraints Editor, SCOPE

Synplify's Constraints Editor allows you to control optimization options and pass timing specifications to the Place and Route software. All timing specifications are passed in the netlist constraints file (NCF) which in turn gets passed directly to the place and route engine. Some of the timing constraints are used by the synthesis engine to produce better synthesis results for the place and route tools.

Clock

Defines a specific clock frequency that overrides the global frequency. Useful when using the CLKDLL in Virtex parts or the DCM in Virtex-II parts.

Inputs/Outputs

Defines the OFFSET IN/OUT constraints for inputs and outputs.

Registers

Constraints for paths feeding into/out of registers.

Multi-cycle Paths

Used to define paths that require multiple number of clocks cycles.

False Paths

Paths that the Timing Analyzer is supposed to ignore.

Attributes

Attributes and synthesis directives that do not fit in one of the above categories (like syn_maxfanout).

Other

Place-and-route tools constraints that are not used by Synplify but will be used by the place-and-route tools.

Using Synplify's Constraints Editor, SCOPE

Xilinx recommends that you let the automatic placement and routing program, PAR, define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place-and-route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a PCB (printed circuit board).

Define the initial pinout by running the place-and-route tools without pin assignments, then locking down the pin placement so that it reflects the locations chosen by the tools. Assign locations to the pins in the Watch design so that the design can function in a Xilinx demonstration board. Because the design is simple and timing is not

critical, these pin assignments do not adversely affect the ability of PAR to place-and-route the design.

For HDL-based designs, these pin assignments can be done in a User Constraints File (.ucf) or with SCOPE. Although UCF files are provided for this tutorial, you will assign some basic timing constraints with a pin location constraint in SCOPE.

To add clock and location constraints:

1. Double-click Edit Constraint File in the Processes for Current Source window.

You will see that Synplify launches with the SCOPE window open.

2. In SCOPE, click the Clocks tab.
3. Now, call up the RTL viewer by selecting **HDL Analyst** → **RTL** → **Hierarchical View**.

Alternatively you can get to the RTL viewer by selecting the XOR symbol on the tool bar in Synplify.

4. Arrange the windows using the zoom functions from within Synplify so that you can see both the RTL view and SCOPE. The zoom functions for the RTL viewer are represented as magnifying glasses on the tool bar in Synplify.
5. Locate the clock net that exits from the BUFG. See [Figure 1-21](#).
6. Click the clock net to highlight it and drag it to the first cell under the Clock column in the Clocks tab.

What you should see now is n:clk_dcm. The 'n' signifies net, and 'clk_dcm' is the name of the net.

7. Insert 50 MHz (or 20 ns) as the desired clock frequency.

Note: Because we are using a DCM, the frequency constraint that was entered at the beginning did not propagate forward to the clock pins of the flip-flops.

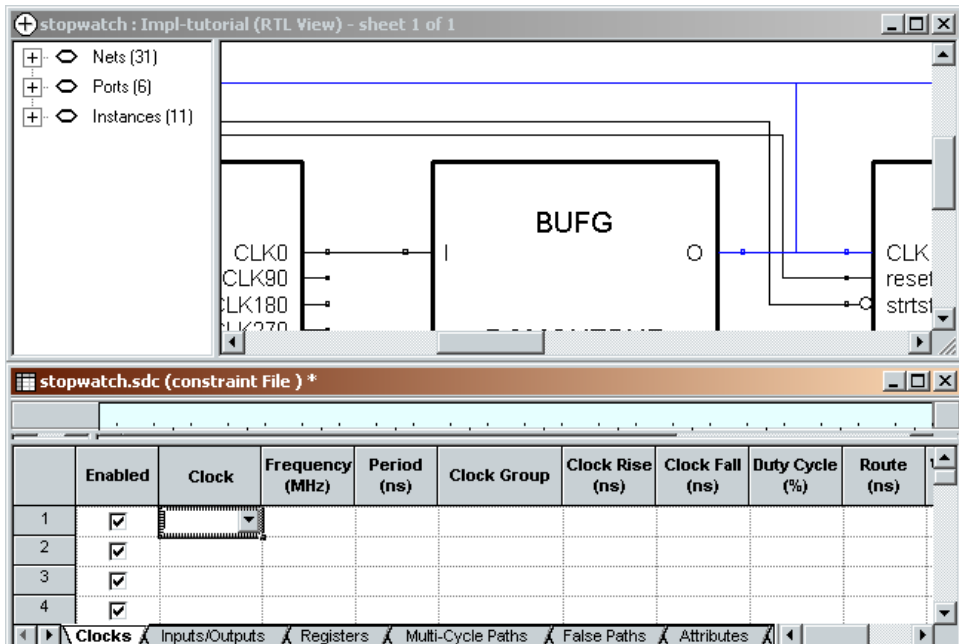


Figure 1-21 Setting a Clock Constraint in SCOPE

8. In SCOPE, click the Attributes tab.
9. Select CLK in the Object column.
10. In the Attribute column, select the Synplify constraint xc_loc.
11. In the Value column, which represents the pin location, select P15. To reassign, click the box and enter the pin number.

See the results in the [Figure 1-22](#).

	Enabled	Object Type	Object	Attribute	Value	Val Type	Description	
1	<input checked="" type="checkbox"/>	port	CLK	xc_loc	P15	string	Port placement	
2	<input checked="" type="checkbox"/>							
3	<input checked="" type="checkbox"/>							
4	<input checked="" type="checkbox"/>							
5	<input checked="" type="checkbox"/>							
6	<input checked="" type="checkbox"/>							
7	<input checked="" type="checkbox"/>							

Navigation: Clocks / Inputs/Outputs / Registers / Multi-Cycle Paths / False Paths / **Attributes**

Figure 1-22 Setting a Location Constraint in SCOPE

12. Save the constraints in Synplify by selecting **File** → **Save** or by clicking the Save icon in the tool bar.
13. Exit Synplify.

Examining Synthesis Results

To view overall synthesis results, double-click on View Map Report under the Mapping process. The report consists of the following three sections:

- “Compiler Summary”
- “Timing Report”
- “Mapping Report”

Compiler Summary

The compiler summary reports on the files in the project. This section of the report is where you will find the errors and warnings associated with each file.

Note: Black boxes (modules not read into Synplify’s design environment) are always noted as Unbound in the Synplify reports. As long as the underlying netlist (.xnf, .ngo, .ngc or .edn) for a black box exists in the project directory, the Implementation tools merge the netlist in during the Translate phase. Since the Tenths module was built using CORE Generator called from the project, the tenths EDN file can be found.

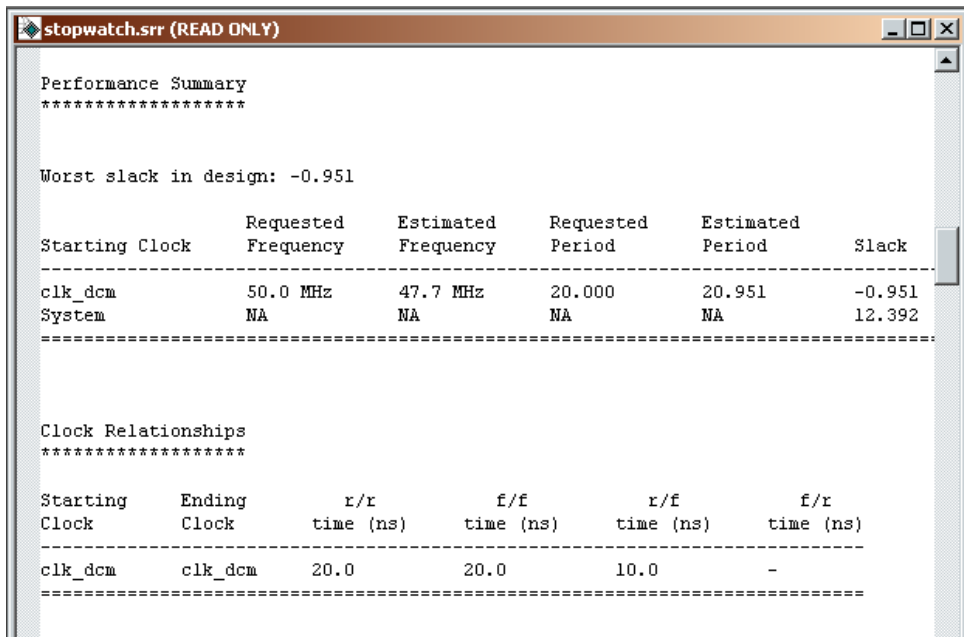


Figure 1-23 Synplify's Estimated Timing Data

Timing Report

The timing report section details information on the constraints that you have entered along with delays on portions of the design that had no constraints. The delay values are based on wireload models and, therefore, must be considered preliminary. Consult the post-route timing reports for the most accurate delay information.

Mapping Report

The mapping report lists all of the components that were used for the design (LUTs, flip-flops, block RAMs, etc.).

At this point, an EDN file exists for the Stopwatch design.

- To perform a pre-synthesis simulation of this design, see the [“Behavioral Simulation” chapter](#).
- To place and route the design, see the [“Design Implementation” chapter](#).
- To perform post-place and route simulation, see the [“Timing Simulation” chapter](#).

Synthesizing the Design using Leonardo Spectrum

Now that the design has been entered and analyzed, the next step is to synthesize the design. In this step, the HDL files are translated into gates and optimized to the target architecture.

1. Set the global synthesis options by selecting stopwatch.vhd (or stopwatch.v).
2. Right-click on the Synthesis process and select Properties.
3. Set the Default Frequency to 50MHz under the Synthesis Options tab.
4. Make sure the Do Not Write NCF box is unchecked under the Netlist Options tab.
5. Click **OK** to accept these values.
6. Select stopwatch.vhd (or stopwatch.v) and double-click on the Synthesize process in the Process Window.

This step can also be done by selecting stopwatch.vhd (or stopwatch.v), and then selecting the Synthesize process in the Processes for Current Source Window.

7. Select **Process** → **Run**.

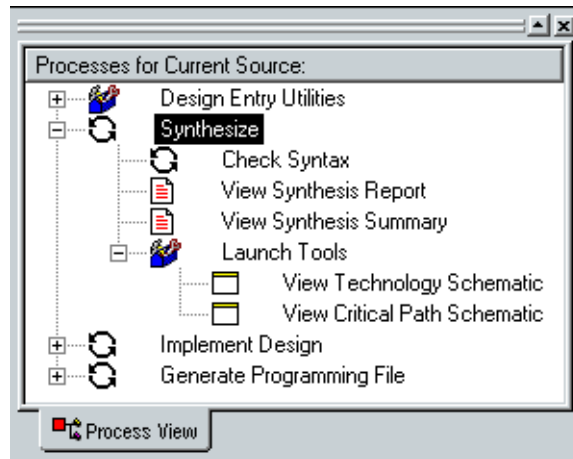


Figure 1-24 Leonardo Spectrum Synthesis/Implementation Window

8. Double-click View Synthesis Report and View Synthesis Summary to see the details of the synthesis.

The Synthesis Report summarizes the compilation, the mapping and the timing of the design. The Synthesis Summary goes into more detail on the mapping and timing of the design.

Schematic-Based Design

This chapter guides you through a typical FPGA schematic-based design procedure using a design of a runner's stopwatch. The design example used in this tutorial demonstrates many device features, software features, and design flow practices that you can apply to your own design. The Watch design targets a Virtex-II device; however, all of the principles and flows taught are applicable to any Xilinx device family, unless otherwise noted.

For an example of how to design with CPLDs, see the online help by selecting **Help** → **ISE Help Contents** in Project Navigator. In the Help Contents under Tutorials, select CPLD Design Flows.

This chapter is the first in the [“Schematic Design Flow.”](#) In the first part of the tutorial, you will use the ISE design entry tools to complete the design. The design is composed of schematic elements, a state machine, a CORE Generator component, and an HDL macro. After the design is successfully entered in the Schematic Editor, you are ready to perform a behavioral simulation with ModelSim, found in the [“Behavioral Simulation”](#) chapter, implementation with the Xilinx Implementation Tools, found in the [“Design Implementation”](#) chapter, and timing simulation with ModelSim, found in the [“Timing Simulation”](#) chapter.

This chapter includes the following sections.

- [“Getting Started”](#)
- [“Overview of Project Navigator”](#)
- [“Overview of Synthesis Tools”](#)
- [“Design Description”](#)
- [“Design Entry”](#)

Getting Started

The following sections describe the basic requirements for running the tutorial.

Required Software

The Xilinx Series ISE package, is required to perform this tutorial. For this design you must install the Virtex-II libraries and device files, and you must be licensed for FPGA Express or Base Express. You must also have the Watch projects on your computer, which may be downloaded from <http://support.xilinx.com>.

Note: An Express license is required to access the Express Constraints GUI.

Installing the Tutorial

This tutorial assumes that the software is installed in the default location, c:\xilinx. If you have installed the software in a different location, substitute your installation path for c:\xilinx.

Unzip the tutorial projects in the c:\xilinx directory, and replace any existing files. The files downloaded from the web have the most recent updates.

Note: For detailed instructions, refer to the *Series ISE 4.1i Install and Release Document*.

Tutorial Project Directories and Files

The following schematic project directories can be downloaded and installed with the tutorial.

- `c:\xilinx\iseexamples\wtut_sc`
(incomplete schematic tutorial)
- `c:\xilinx\iseexamples\watch_sc`
(complete schematic tutorial)

The schematic tutorial files are copied into these directories.

The `wtut_sc` project contains an incomplete copy of the tutorial design. You will create the remaining files when you perform the tutorial. As described in a later step, you can copy this project to another area and perform the tutorial in this new area if desired.

The `watch_sc` solution project contains the design files for the completed tutorial including schematics and the bitstream file. To conserve disk space, some intermediate files are not provided. Do not overwrite any files in the solutions directories.

Copying the Tutorial Files (Optional)

You can either work within the project directory as it has been downloaded, or you can make a copy to work on. To make a working copy of the tutorial files, use Windows Explorer to copy the `wtut_sc` directory to another location. The `wtut_sc` project directory contains all of the necessary project files.

Starting the ISE Software

To follow along with this tutorial, you will need to launch the ISE software package. To do so:

1. Double-click the ISE Project Navigator icon on your desktop or select **Start** → **Programs** → **Xilinx ISE 4.1i** → **Project Navigator**.



2. From Project Navigator, select **File** → **Open Project**.

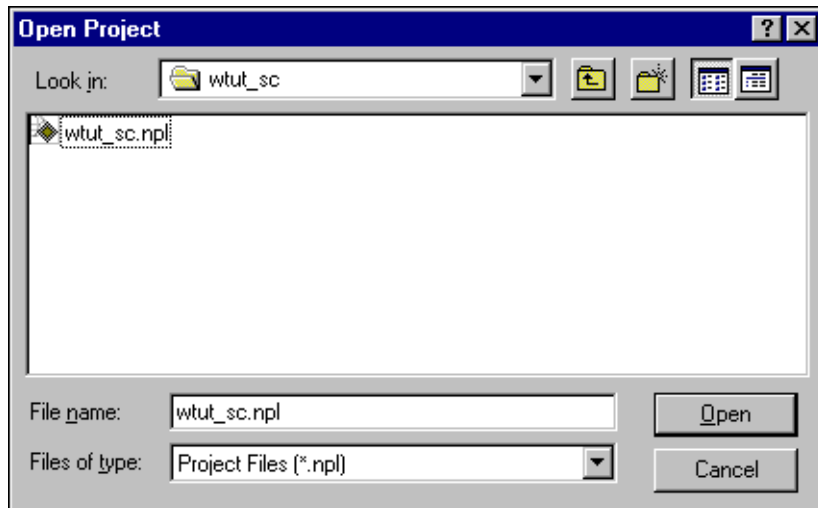


Figure 2-1 Getting Started Dialog Box

3. In the Directories list, browse to `c:\xilinx\iseexamples\wtut_sc`.
4. Double-click `wtut_sc.npl`.

Overview of Project Navigator

Project Navigator controls all aspects of the design flow. Through Project Navigator, you can access all of the various design entry and design implementation tools. You can also access the files and documents associated with your project. Project Navigator maintains a flat directory structure, therefore you must maintain revision control through the use of snapshots.

Project Navigator is divided into four main subwindows. On the top left is the Sources in Project window, which hierarchically displays the elements included in the project. Beneath the Sources in Project window is the Process Window, which displays available processes. The third window at the bottom of Project Navigator is the Message Console, which shows status, error, and warning messages. It is updated during all project actions. On the right, the fourth window is the HDL Editor. From this window you can edit source files and access the Language Templates. These windows are discussed in more detail in the following sections.

Sources in Project Window

The Sources in Project window has three tabs that provide information for the user. Each tab is discussed in further detail below.

Module View

In the Module View tab of the Sources in Project window, user documents, part types, synthesis tools, and design source files are displayed. User documents are listed under the project name. Source files are listed under the part name and synthesis tool. Next to each filename is an icon that tells you the file type (HDL file, schematic, core, text file, for example). If a file contains lower levels of hierarchy, the icon has a + to the left of the name. HDL files have this + to show the entities (VHDL) or modules (Verilog) within the file. You can expand the tree by clicking this icon. You can open a file for editing by double-clicking on the filename.

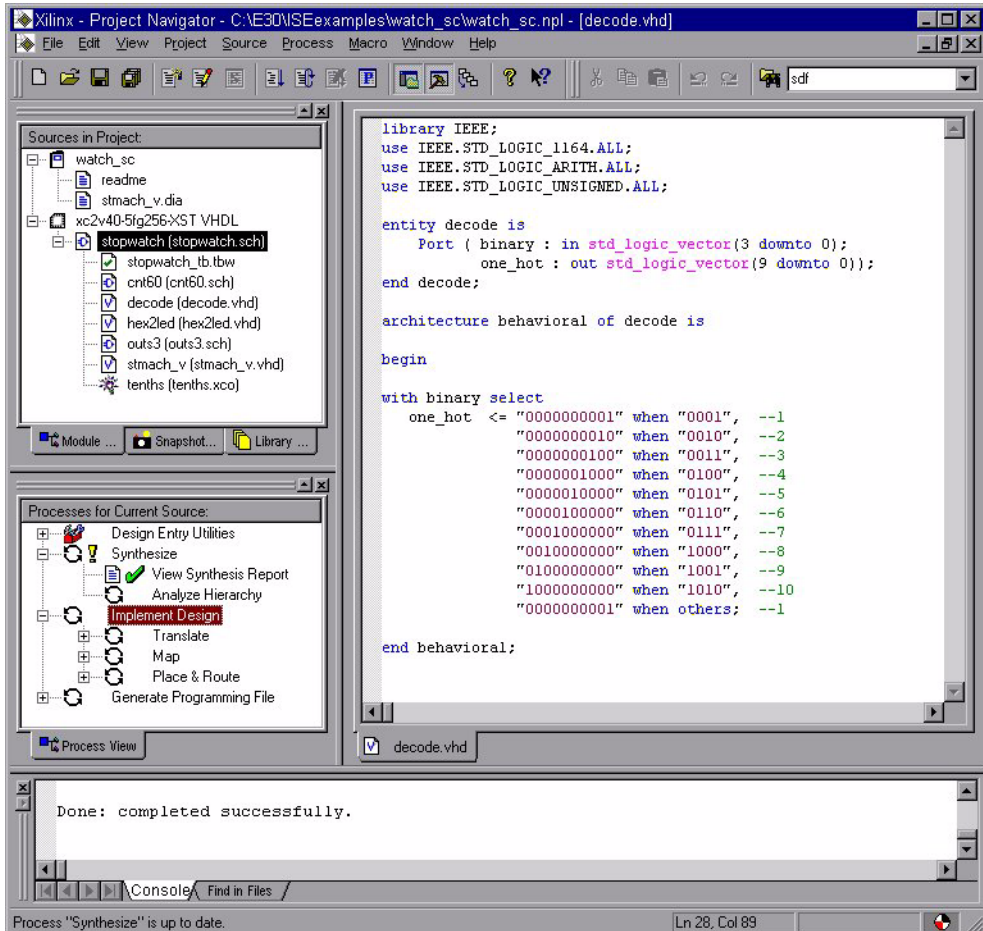


Figure 2-2 Project Navigator

Snapshot View

A snapshot is a method of revision control. At any time in the design cycle you can take a snapshot. A snapshot consists of all files in the current working directory. This also includes synthesis and simulation subdirectories. You can restore a snapshot to resume work at a particular phase in the design cycle. In the Snapshot View tab of the Sources in Project window, all of the snapshots associated with the open project are displayed. This allows you to view the reports,

user documents, and source files. All information displayed in the snapshot view is read-only.

Note: Remote sources are not copied with the snapshot. A reference to the remote source is maintained in the snapshot.

Library View

In the Library View tab of the Sources in Project window, all libraries associated with the project are displayed.

Processes for Current Source Window

Process View

The Process Window is located beneath the Sources in Project window. This window is context sensitive and changes based upon the selected source. The status of each process is displayed on the process icon as a red x, yellow exclamation, or green check mark. The Process Window provides access to the following functions:

- Design Entry Utilities—Provides access to symbol generation, test bench generation, and instantiation templates.
- Synthesis—Provides access to check syntax, synthesis, and synthesis reports. This varies depending on the synthesis tools you use.
- Implement Design—Provides access to implementation tools, design flow reports, and point tools.
- Create Programming File—Provides access to the configuration tools and bitstream generation.

The Processes for Current Source window incorporates automake technology. This means you can select any process in the flow, and the software automatically runs the processes necessary to get to the step you selected. For example, if the synthesis process has not been run, it is not necessary to run the synthesis process before running the implementation process. Running the implementation process causes the synthesis process to run before implementation runs.

Console Window

Error, warning, and informational messages are displayed in the Console Window. Errors and warnings are signified by a red box next to the message, while warnings have a yellow box.

Error Navigation to Source

The Console Window provides the ability to navigate from a synthesis error or warning message to the source HDL file. This can be done by right-clicking the error or warning message and selecting, Goto Source. This opens the HDL source file and move the cursor to the line with the error.

Error Navigation to Solution Record

If an error or warning message has a solution record associated with it, the Console Window provides the ability to navigate from the error or warning message to the solution record on the support.xilinx.com web site. These type of errors or warnings can be identified by the web icon to the left of the error. To navigate to the solution record, right-click the error or warning message and select Goto Solution Record. This opens a web browser and display any solution records applicable to this message.

Snapshots

Snapshots enable you to maintain revision control over the design. A snapshot contains all of the files in the project directory.

Creating a Snapshot

You can create a snapshot by selecting **Project → Take a Snapshot**. This opens the Take a Snapshot of the Project dialog box. Enter the snapshot name and any comments associated with the snapshot. The snapshot contains all of the files in the project directory along with project settings.

Restoring a Snapshot

The Snapshot View, of the Sources in Project window, contains a list of all the snapshots available in the current project. Since snapshots are read-only, a snapshot must be restored in order to work with it. To

do this select the snapshot and select **Project** → **Replace with Snapshot**. You will be prompted to create a snapshot of the current project directory and restore the selected snapshot for further work.

Viewing a Snapshot

The Snapshot View of the Sources in Project window contains a list of all the snapshots available in the current project. You can open a snapshot to review report or verify process status by right-clicking the snapshot and selecting Open.

Overview of Synthesis Tools

The following synthesis tools can be used in a schematic-based design. This section lists the devices supported by each synthesis tool and some process properties information.

Xilinx Synthesis Technology (XST)

Supported Devices

XST supports the following devices.

- Virtex™ / -E / -II / - II Pro
- Spartan™-II / -IIE
- XC9500™ /XL /XV
- Coolrunner™ /-II

Process Properties

Process properties allow you to control the synthesis results of XST. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties you can control the synthesis results for area or speed, and the amount of time the synthesizer runs.

More detailed information is available in the *XST User Guide*.

FPGA Express

Supported Devices

FPGA Express supports the following devices.

- Virtex™ / -E / -II / -II Pro
- Spartan™ / XL / -II / -IIE
- XC9500™ / XL / XV
- XC4000™ E / EX / XL / XV / XLA
- Coolrunner™ / -II

Process Properties

Process properties allow you to control the synthesis results of FPGA Express. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties you can control the synthesis results for area or speed, and the amount of time the synthesizer runs.

More detailed information is available in the FPGA Express online help.

Leonardo Spectrum

This synthesis tool is not part of the ISE package, therefore it is not available unless you have purchased it separately.

Supported Devices

Leonardo supports the following devices.

- Virtex™ / -E / -II / -II Pro
- Spartan™ / XL / -II / -IIE
- XC9500™ / XL / XV
- XC4000™ E / EX / XL / XV / XLA
- Coolrunner™ / -II

Process Properties

Process properties allow you to control the synthesis results of Leonardo Spectrum. Two commonly used properties are Optimization Goal and Optimization Effort. Through these properties you can control the synthesis results for area or speed and the amount of time the synthesizer runs.

More detailed information is available in the Leonardo Spectrum online help.

Design Description

Throughout this tutorial, the design is referred to as Watch.

The design used in this tutorial is a hierarchical, schematic-based design, meaning that the top-level design file is a schematic sheet that refers to several other lower-level macros. The lower-level macros are a variety of different types of modules including schematic-based modules, CORE Generator modules, state machine modules, and HDL modules.

The design begins as an unfinished design. Throughout the tutorial, you will complete the design by creating some of the modules, and by completing some others from existing files. After the design is complete, you will simulate the design to verify its functionality.

Watch is a simple runner's stopwatch. The completed schematic is shown in the following figure.

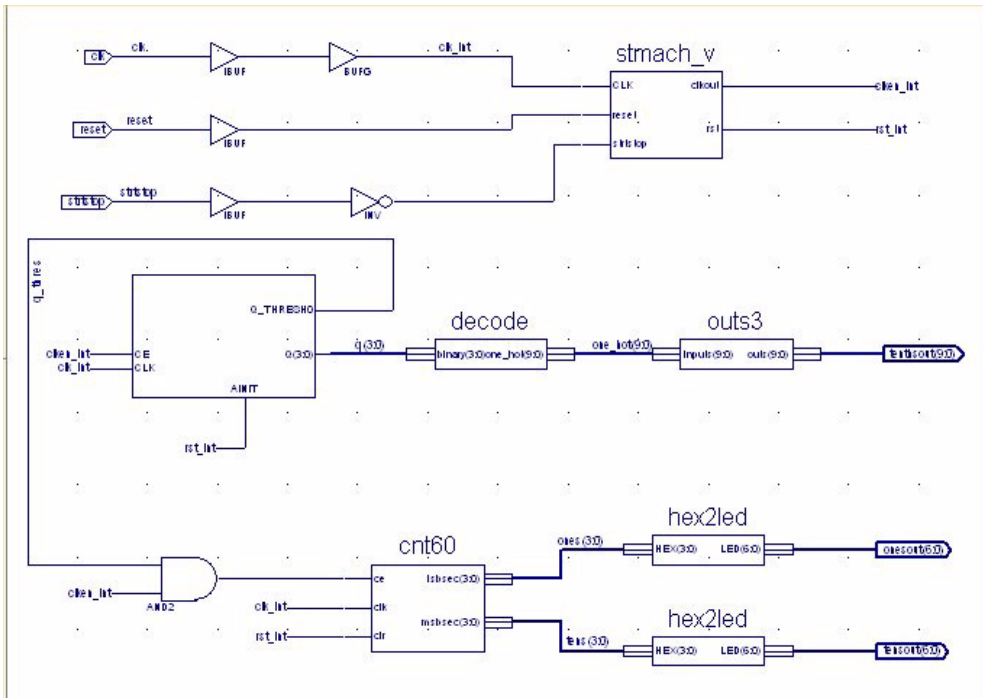


Figure 2-3 Completed Watch Schematic

There are three external inputs and three external outputs in the completed design. The following list summarizes the inputs and outputs, and their respective functions.

Inputs:

- **CLK**
System clock for the Watch design.
- **STRTSTOP**
Starts and stops the stopwatch. This is an active-low signal that acts like the start/stop button on a runner's stopwatch.
- **RESET**
Resets the stopwatch to 00.0 after it has been stopped.

Outputs:

- TENSOUT(6:0)
7-bit bus that represents the *tens* digit of the stopwatch value. This bus is in 7-segment display format to be viewable on the 7-segment LED display.
- ONESOUT(6:0)
Similar to the TENSOUT bus above, but represents the *ones* digit of the stopwatch value.
- TENTHSOUT(9:0)
10-bit bus which represents the *tenths* digit of the stopwatch value. This bus is one-hot encoded.

The completed design consists of the following functional blocks. Most of these blocks do not yet appear on the schematic sheet in the tutorial project since they are created during this tutorial.

Functional Blocks

- STMACH_V
State Machine macro. This module uses StateCAD to enter and implement the state machine.
- CNT60
Schematic-based module which counts from 0 to 59 decimal. This macro has two 4-bit outputs, which represent the *ones* and *tens* digits of the decimal values, respectively.
- TENTHS
CORE Generator 4-bit, binary encoded counter. This macro outputs a 4-bit code that is decoded to represent the *tenths* digit of the watch value as a 10-bit one-hot encoded value.
- HEX2LED
HDL-based macro. This macro decodes the *ones* and *tens* digit values from hexadecimal to 7-segment display format.

- **OUTS3**
Schematic based macro containing inverters.
- **DECODE**
Decodes the CORE Generator output from 4-bit binary to a 10-bit one-hot output.

Design Entry

In this hierarchical design, you will create various types of macros, including schematic-based macros, HDL-based macros, state machine macros, and CORE Generator macros. You will learn the process for creating each of these types of macros, and connect them together to create the completed Watch design. This tutorial gives you experience with creating and using each type of design macro so that you can apply this knowledge to your own design.

Starting the Schematic Editor

There are two different ways to open the Schematic Editor tool.

- Select **Project** → **New Source**. This will open the New Source dialog box where the schematic source type can be selected. This will create a new schematic.
- or
- Double-click the file name stopwatch.sch in the Sources in Project window.

The Schematic Editor opens with the Watch schematic sheet loaded. The Watch schematic is incomplete at this point. Throughout the tutorial, you will create the components to complete the design. The unfinished design is shown in the figure below.

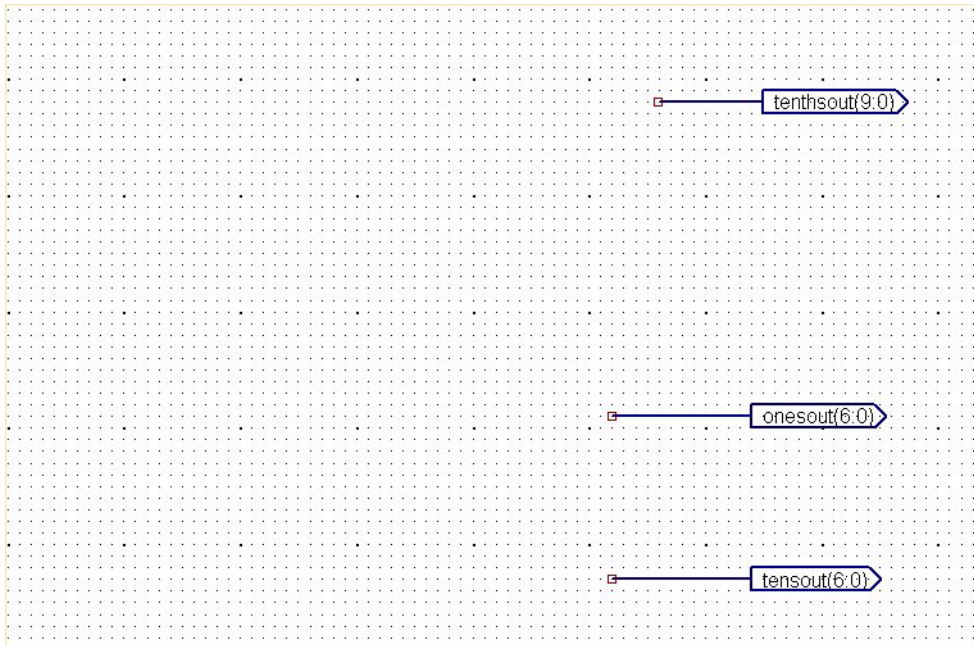


Figure 2-4 Incomplete Watch Schematic

If you need to stop the tutorial at any time, save your work by selecting **File** → **Save**.

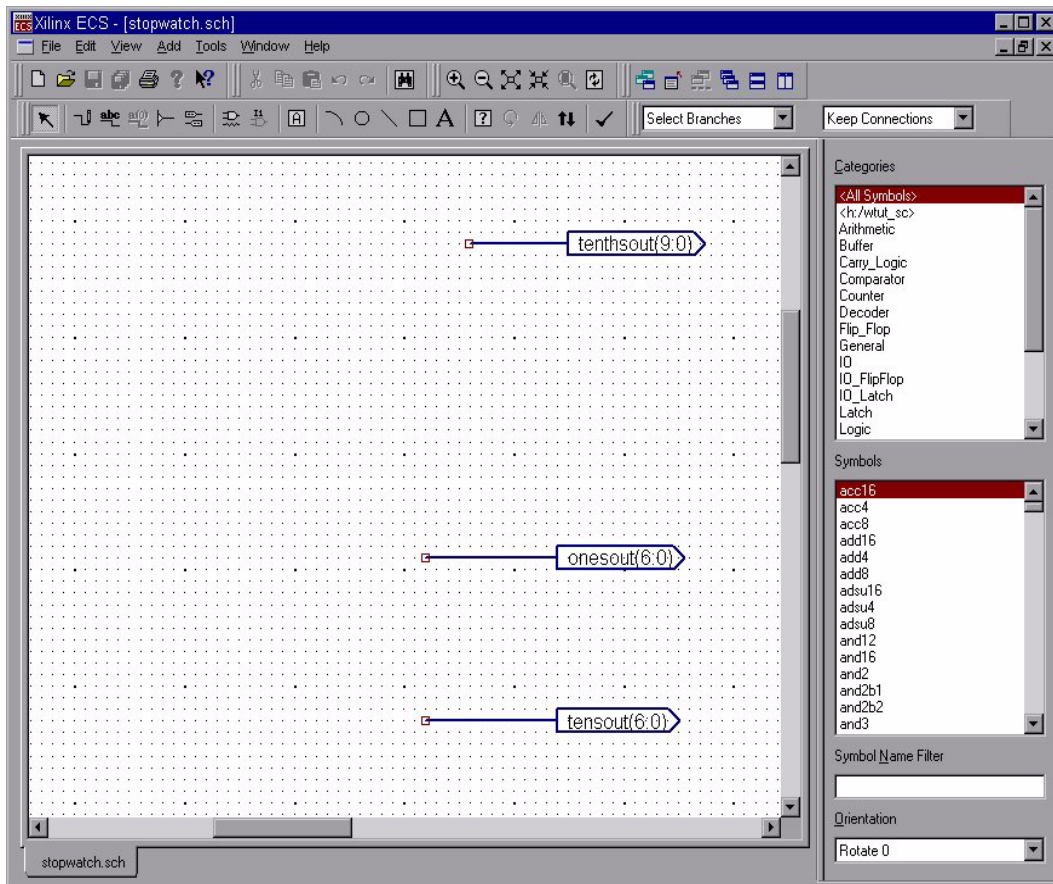


Figure 2-5 Schematic Editor

Manipulating the Screen View

Under the View pulldown menu is a series of commands that modify the viewing area of the Schematic Editor window. Select **View** → **Zoom** → **In** until you can comfortably view the schematic.

Creating a Schematic-Based Macro

A schematic-based macro consists of a symbol and an underlying schematic. You can create either the underlying schematic or the symbol first, and the tools can automatically generate the corresponding symbol or schematic file, respectively. In the following steps, you will create a schematic-based macro by using the New Source Wizard. A template schematic file is then created by the tools, and you complete the schematic with the appropriate logic. The created macro is then automatically added to the project's library.

The macro you will create is called CNT60. CNT60 is a binary counter with two 4-bit outputs, which represent the Ones and Tens values of the stopwatch. The counter counts from 0 to 59, decimal.

1. Select **Project** → **New Source** in the Project Navigator. The New Source dialog opens (see [Figure 2-6](#)).

The New Source dialog provides a list of all available source types.

2. Select Schematic as the source type.
3. Enter 'CNT60' as the file name.
4. Click **Next**, then **Finish**.

This creates a new schematic named CNT60 and adds the schematic file to the project.

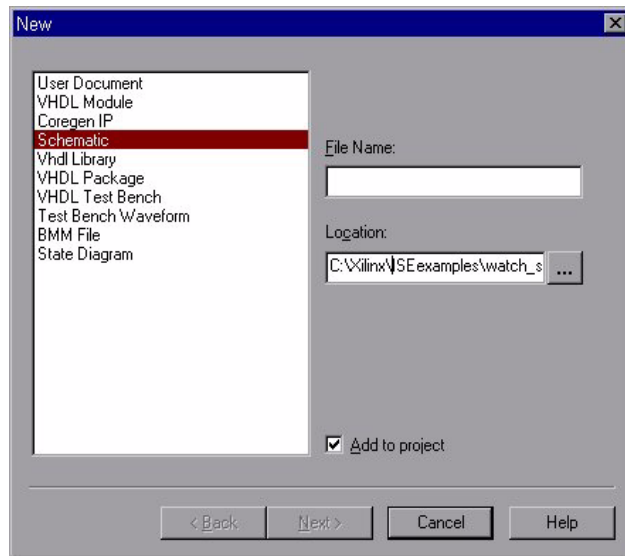


Figure 2-6 New Source Dialog Box

Creating the CNT60 Schematic

You have now created an empty schematic for CNT60. The next step is to add the components which comprise the CNT60 macro. You can then reference this macro symbol by placing it on a schematic sheet.

Connectivity—I/O Markers

I/O markers logically connect the CNT60 symbol and its underlying schematic. The name of each pin on the symbol must have a corresponding connector in the underlying schematic.

To add hierarchy connectors manually, select **Add → Marker** or click the I/O Marker icon from the Tools toolbar.



Note: The Tools toolbar is displayed by selecting **View → Toolbars**, and clicking the Tools toolbar option.

When you save a macro, the Schematic Editor checks the I/O markers against the corresponding symbol. If there is a discrepancy, you can

let the software update the symbol automatically, or you can modify the symbol manually. I/O markers should be used to connect signals between levels of hierarchy and also to specify the ports on top-level schematic sheets.

Project Libraries

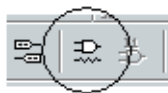
ISE contains several different types of libraries.

- Schematic: Xilinx Unified Symbol libraries contain macros and primitives. Engineering Capture System(ECS) contains access to these library elements using the Symbol Library dialog box.
- Schematic: Project specific libraries are a collection of the symbols created for the current project. Each symbol is maintained as a sym file in the current project directory.
- Simulation: Different libraries are utilized for functional and timing simulation. A discussion of each library type can be found in the Function Simulation and Timing Simulation chapters.

Adding Components to CNT60

Components from the device and project libraries for the given project are available from the Symbol Libraries toolbox to place on the schematic. The available components listed in this toolbox are arranged alphabetically within each library.

1. From the menu bar, select **Add** → **Symbol** or click the Add Symbol icon from the Tools toolbar.



This opens the Symbol Browser window, if it is not already shown to the right of the schematic editor, and displays the libraries and their corresponding components.

Note: Components can be rotated in two ways. New components being added to a schematic can be rotated by selecting CTRL+R. Existing components can be rotated by selecting the move mode, selecting the component, and then selecting CTRL+R.

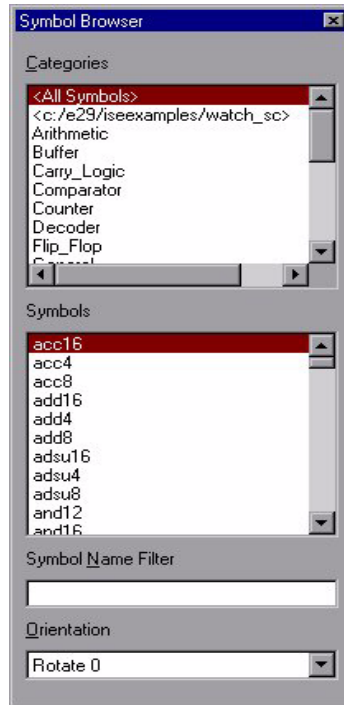


Figure 2-7 Symbol Browser Dialog Box

The first component you will place is an AND2, a 2-input AND gate.

2. Select this component one of two ways:
 - Highlight the Logic category from the Symbol Browser and select it from the symbols list.
 - Select All Symbols and type 'AND2' in the Symbol Name Filter at the bottom of the Symbol Browser window.
3. Move the mouse back into the schematic window.
4. Move the symbol outline to the location shown in the following figure and click the left mouse button to place the object.

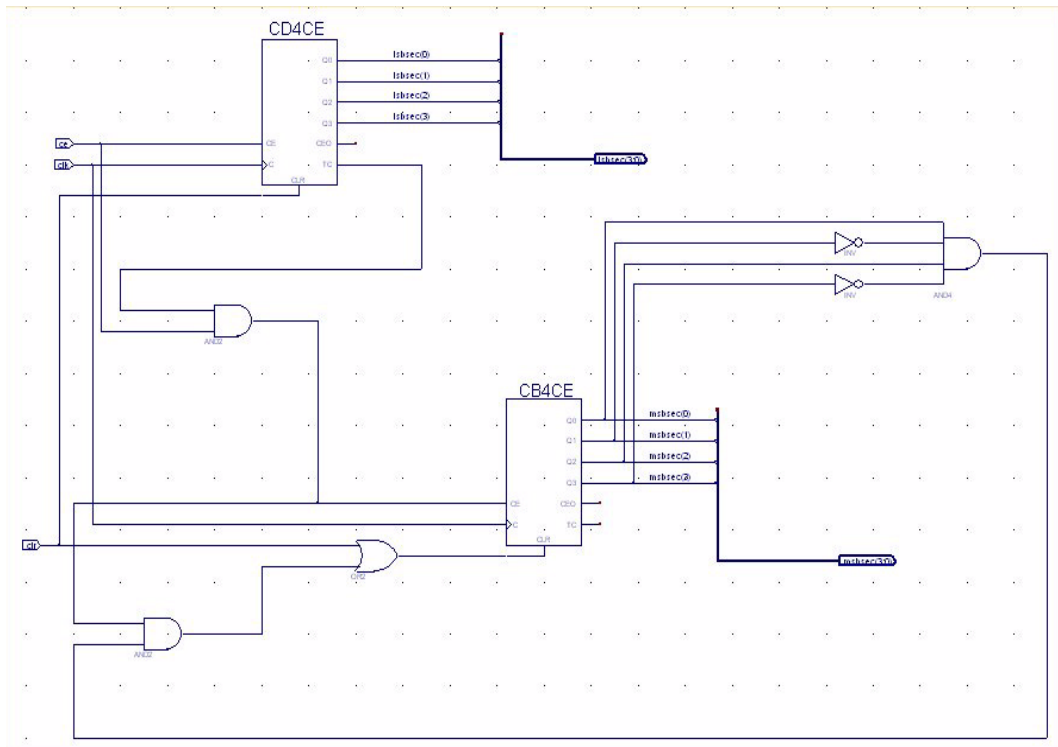


Figure 2-8 Completed CNT60 Schematic

Note: The preceding schematic illustrates the completed CNT schematic. Use this figure as a reference for drawing nets and buses in the following sections.

Correcting Mistakes

If you make a mistake when placing a component, you can easily move or delete the component.

To exit the Symbols Mode, press the **Esc** key on the keyboard.

To delete the component in one of two ways:

- Click the component and press the delete key on your keyboard.
- Right-click the component and click Delete.

To move the component, click the component and drag the mouse around the screen.

Placing the Remaining Components

Follow the steps listed previously in the [“Adding Components to CNT60”](#) section to place the CD4CE, OR2, CB4CE, INV, and AND4 components on the schematic sheet as shown in [Figure 2-8](#). For a detailed description of the functionality of each of these components, refer to the Xilinx *Libraries Guide*.

Drawing Wires

Use the Add Wire icon in the Tools toolbar to draw wires (also called nets) between the various components on the schematic.

Signals can be logically connected by naming multiple segments identically. In this case, the nets do not need to be physically connected on the schematic to make the logical connection. In the CNT60 schematic, you will draw wires to connect the components together. The nets for the LSBSEC and MSBSEC buses will be drawn in the next section.

Perform the following steps to draw a net between the AND2 and the CB4CE components on the CNT60 schematic.

1. Select **Add** → **Wire** or click the Add Wires icon in the Tools toolbar.



Figure 2-9 Add Wires Icon

2. Click the source symbol pin (output pin of the AND2), then click the destination pin (CE pin on the CB4CE). The net will automatically be drawn between the two pins.

Note: To specify the shape of the net, move the mouse in the direction you want to draw the net and then click the mouse to create a 90-degree bend in the wire.

Draw the nets to connect the remaining components as shown in the [Figure 2-8](#). Net names will be added in a later section. To draw a net between an already existing net and a pin, click once on the component pin and once on the existing net. A junction point will be drawn on the existing net.

You should now have all the nets drawn except those connected to the LSBSEC and MSBSEC buses. You will draw these in the next section.

Adding Buses

In ECS, a bus is simply a wire which has been given a multi-bit name. In order to add a bus follow the same methodology for adding wires and then add the proper name. Once a bus has been created, you have the option of “tapping” this bus off to use each signal individually.

In this CNT60 schematic, create two buses called LSBSEC(3:0) and MSBSEC(3:0), each comprised of the 4 output bits of each counter. Then connect an I/O marker to each bus in order to connect them to the CNT60 symbol. The results can be found in the completed schematic.

To add the buses, LSBSEC(3:0) and MSBSEC(3:0), to the schematic, perform the following steps:

1. Select **Add** → **Wire** or click the Add Wires icon in the Tools toolbar.
2. Click to the right of the CB4CE and draw a wire down and to the right of the symbol.
3. Terminate the wire one of two ways:
 - Double-click the mouse.
 - Single-click with the right mouse button.

To change the wire into a bus, the wire must be named.

4. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.



Figure 2-10 Add Net Name Icon

5. With the keyboard enter ‘msbsec(3:0)’ and hit enter.
This will attach the bus name to the cursor.

6. Click the end of the bus to apply the name.
This will change the wire into a bus.
7. To verify this, zoom in. The bus is represented visually by a thicker wire.

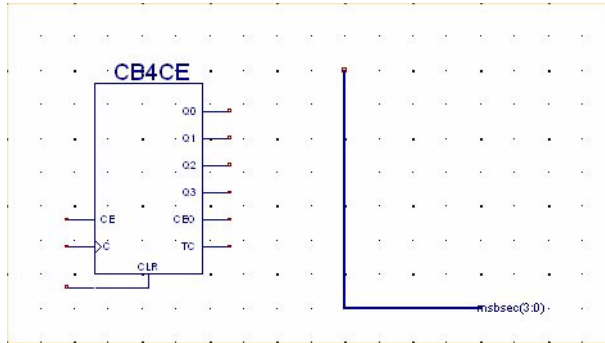


Figure 2-11 Creating a Bus by Name

Repeat Steps 2 through 7 for the LSBSEC(3:0) bus.

8. After adding the two buses, press **Esc** or right-click to exit the Draw Buses mode.

Adding Bus Taps

Next, add nets to attach the appropriate pins from the CB4CE and CD4CE counters to the buses. Use Bus Taps to tap off a single bit of a bus and connect it to another component. The Schematic Capture tool names the bus taps incrementally as they are drawn.

Note: Enlarging the view of the schematic will enable greater precision when drawing the nets.

To tap off a single bit of each bus:

1. Select **Add** → **Bus Tap** or click the Add Bus Tap icon in the Tools toolbar.



Figure 2-12 Add Bus Tap Icon

The cursor changes, indicating that you are now in Draw Bus Taps mode.

2. In the options toolbar, choose the correct orientation for the bus so that the bottom of the triangle is placed on the bus. The other side of the bus should now be pointing to an unconnected pin.

Repeat steps 1 & 2 to tap off the other three bits of the bus.

To connect each of the tap off bits:

1. Select **Add** → **Wire** or click the Add Wire icon in the Tools toolbar.
2. Draw a wire from the other end of the bus taps to the corresponding pins.
3. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.
4. Type in 'msbsec(0)' in the blank area of the options toolbar.

The net name is now at the end of your cursor.

5. Increment subsequent net names by changing Keep Name to Increment Name in the add net name option toolbar.

Note: Alternatively, name the first net msbsec(3) and decrement as nets are named from the bottom up.

6. With the Increment Name option selected, start at the top net and continue clicking down until you have named the fourth and final net msbsec(3).

Repeat Steps 1 through 6 for the lsbsec(3:0) bus.

7. Press **Esc** or right-click to exit the Draw Bus Taps mode.

8. Complete the schematic by drawing the nets to connect the msbsec bus taps to the INV and AND4 components. If necessary, refer to “[Drawing Wires](#)” for guidance.
9. Compare your CNT60 schematic again with the [Figure 2-8](#) to ensure that all connections are made properly.

Note: If the nets appear disconnected, select **View** → **Redraw** to refresh the screen.

Adding Net Names

Next, add net names to the clk, clr, and ce nets.

1. Select **Add** → **Net Name** or click the Add Net Name icon in the Tools toolbar.
2. In the Options toolbar, enter ‘clk’ into the empty box then move the cursor back to the schematic.



Figure 2-13 Options Toolbar for Add Net Name

Note: The Options toolbar changes depending on which tool you have selected in the Tools toolbar.

The net name clk is now attached to the cursor.

3. Click the end of the clk net.

The name is then attached to the end of the net. If the net name is not attached to the end of the net but appears above the net, delete the net name and repeat the process.

4. Repeat steps 1-3 for ce and clr.

Adding I/O Markers

I/O markers are used to determine the ports on a macro or the top level schematic. Add I/O markers to the CNT60 schematic to determine the macro ports.

Follow these steps to add the I/O markers as shown in [Figure 2-8](#).

1. Select **Add** → **I/O Marker**.

The I/O marker dialog box opens.

2. Select input.
3. Click and drag a box around the following nets: clk, ce, and clr.
4. Change the marker type to output in the Options toolbar and add a marker to the msbsec(3:0) and lsbsec(3:0) nets.

Saving the Schematic

The CNT60 schematic is now complete.

Save the schematic by selecting **File** → **Save** or clicking the Save icon in the toolbar.



Creating the CNT60 symbol

The symbol representing the CNT60 schematic will now be created in Project Navigator.

1. In the Sources in Project window, select cnt60.sch.
2. In the Processes for Current Source window, click the + beside Design Entry Utilities to expand the hierarchy.
3. Double-click Create Schematic Symbol.

Placing the CNT60 Macro

So far, you have created the CNT60 macro. The next step is to place this macro on the top-level Watch schematic sheet, where it may then be connected to other components in the design.

1. Open the Watch schematic sheet by double-clicking stopwatch.sch in the Sources in Project window.
2. Select the Add Symbol icon to open the Symbol Browser dialog box.



Figure 2-14 Add Symbol Icon

3. Select the Local Symbols library and find the newly created CNT60 macro in this list. Select this component.
4. Place the CNT60 macro as shown below.

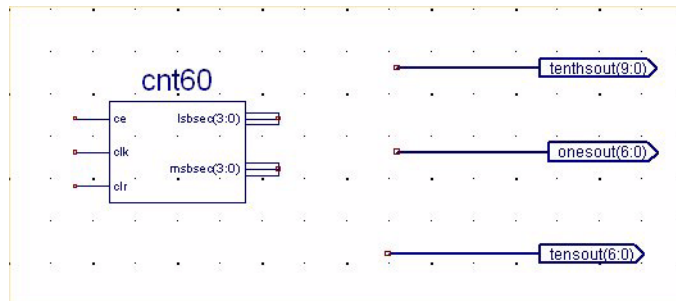


Figure 2-15 Placing the CNT60 Macro

5. Right-click to release this symbol and select another. Notice that the Symbol Browser window remains open if floating. With this window open, you can quickly place additional symbols without having to click the Add Symbol icon again.
6. To close the Symbols window, click the X in the upper right corner of the window or redock it to the right of the application.

Note: Do not worry about connecting nets to the pins of the CNT60 symbol. You will do this after adding other components to the Watch schematic.

Creating a CORE Generator Module

CORE Generator is a graphical interactive design tool you use to create high-level modules such as counters, shift registers, RAM and multiplexers. You can customize and pre-optimize the modules to take advantage of the inherent architectural features of the Xilinx FPGA architectures, such as Fast Carry Logic for arithmetic functions, and on-chip RAM for dual-port and synchronous RAM.

In this section, create a CORE Generator module called Tenthths. Tenthths is a 4-bit binary encoded counter. The 4-bit number is then decoded to count the tenths digit of the stopwatch's time value.

Creating the Core Generator module

Select the type of module you want and the specific features of the module in the CORE dialog box. To invoke this dialog box:

1. In Project Navigator, select **Project** → **New Source**.
2. Select Coregen IP, enter 'tenths' in the File Name field.
3. Click **Next** and then **Finish**.

The Xilinx CORE Generator 4.1i opens and displays a list of possible COREs available.

4. Double-click **Basic Elements - Counters**.
5. Double-click **Binary Counter** to open the Binary Counter dialog box. This dialog box allows you to customize the counter to the design specifications.
6. Fill in the Binary Counter dialog box with the following settings:
 - **Component Name:** tenths
Defines the name of the module.
 - **Output Width:** 4
Defines the width of the output bus.
 - **Operation:** Up
Defines how the counter will operate. This field is dependant on the type of module you select.

- Count Style: Count by Constant
Allows counting by a constant or a user supplied variable.
 - Count Restrictions: Enable and Count To Value A (HEX)
This dictates the maximum count value.
 - Threshold Options: Threshold 0 set to A
Signal goes high when the value specified has been reached.
 - Threshold Options: Registered
7. Click the **Register Options** button to open the Register Options dialog box.
 8. Enter the following settings.
 - Clock Enable: Selected
 - Asynchronous Settings: Init with a value of 1.
 - Synchronous Settings: None
 9. Click **OK**.
 10. Check that *only* the following pins are used.
 - AINIT
 - CE
 - Q
 - Q_Thresh0
 - CLK

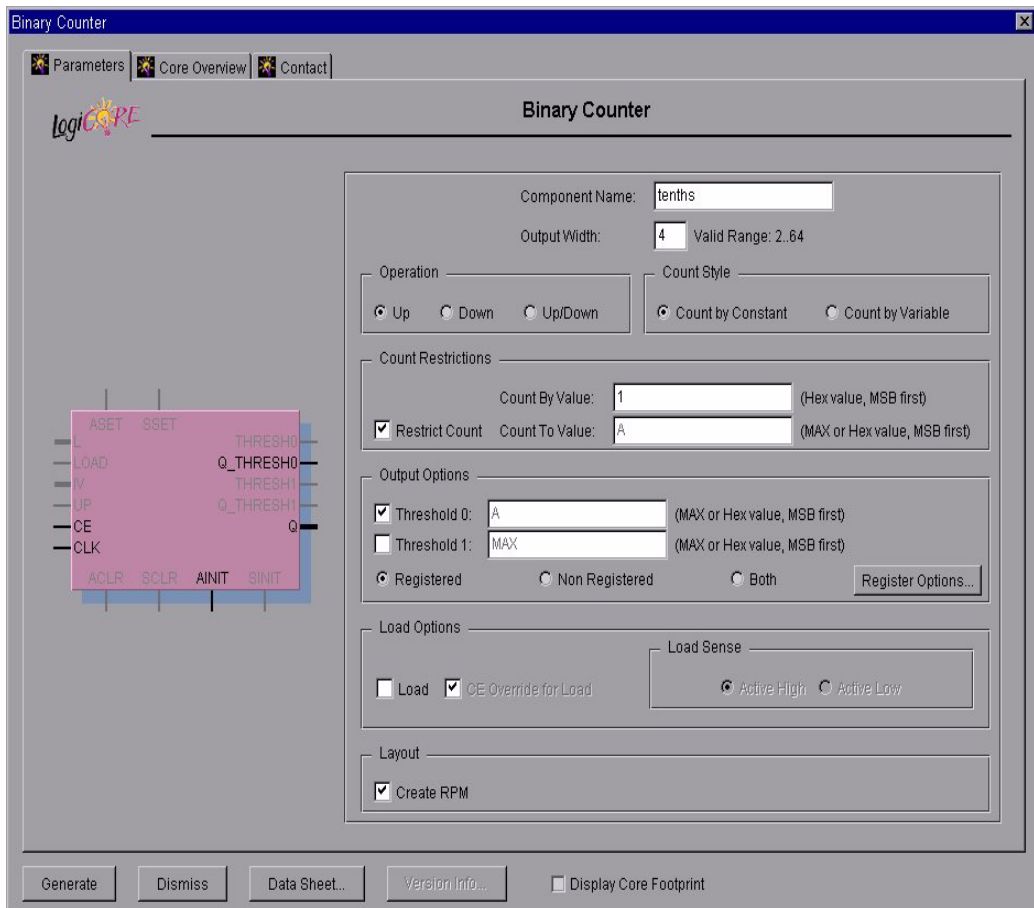


Figure 2-16 CORE Generator Module Selector

11. Click **Generate**.

The module is created and automatically added to the project library.

Note: A number of files are added to the project directory. These files are:

- tenths.edn

This file is the netlist that is used during the Translate phase of implementation.

- tenths.vhd or tenths.v

This is the instantiation template that is used to incorporate the CORE Generator module in your source HDL.

- tenths.xco

This file stores the configuration information for the Tenth module and is used as a project source.

- coregen.prj

This file stores the Coregen configuration for the project.

12. Select **Dismiss** and close CORE Generator.

Creating a State Machine Module

With VSS State CAD, you can graphically create finite state machines: states, inputs/outputs, and state transition conditions. Transition conditions and state actions are typed into the diagram using language independent syntax. The State Editor then exports the diagram to either VHDL, Verilog or ABEL code. The resulting HDL file is finally synthesized to create a netlist and/or macro for you to place on a schematic sheet.

For this tutorial, a partially complete state machine diagram is provided. In the next section, complete the diagram and synthesize the module into a macro to place on the Watch schematic. A completed VHDL State Machine diagram has been provided for you in watch_sc.

Opening the State Editor

You can invoke VSS State CAD from Project Navigator New Source Wizard for new diagrams. The tutorial utilizes an existing diagram which you will complete.

To open the diagram, double-click stmach_v.dia, which opens State CAD and loads state machine.

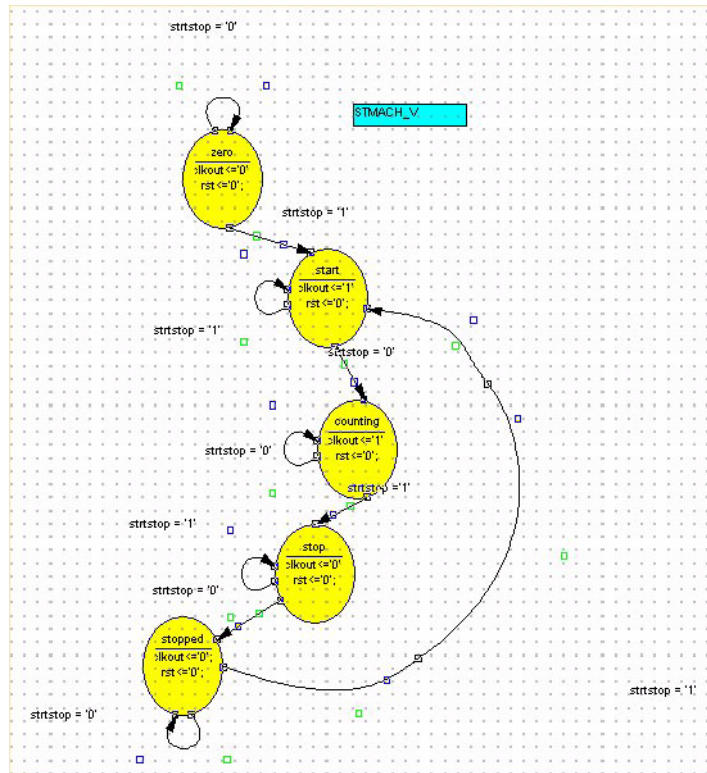


Figure 2-17 Incomplete State Machine Diagram

In the incomplete state machine diagram above:

- The circles represent the various states.
- The black expressions are the transition conditions, defining how you move between states.
- The output expressions for each state are contained in the circle representing the state.

In the state machine diagrams, the transition conditions and the state actions are written in language independent syntax and then exported to Verilog, VHDL, or ABEL.

In the following section, add the remaining states, transitions, actions, and a reset condition to complete the state machine.

Adding New States

Complete the state machine by adding a new state called CLEAR.

1. Click the Draw State icon in the vertical toolbar.



The state bubble is now attached to the cursor.

2. Place the new state on the left-hand side of the diagram as shown [Figure 2-18](#). Click the mouse to place the state bubble.
3. The state is given a default name, in this case STATE0. Double-click the STATE0 in the state bubble, and change the name of the state to **CLEAR**.

Note: The name of the state is for your use only; it does not affect the synthesis, and so you can name it whatever you want.

4. Click **OK**.

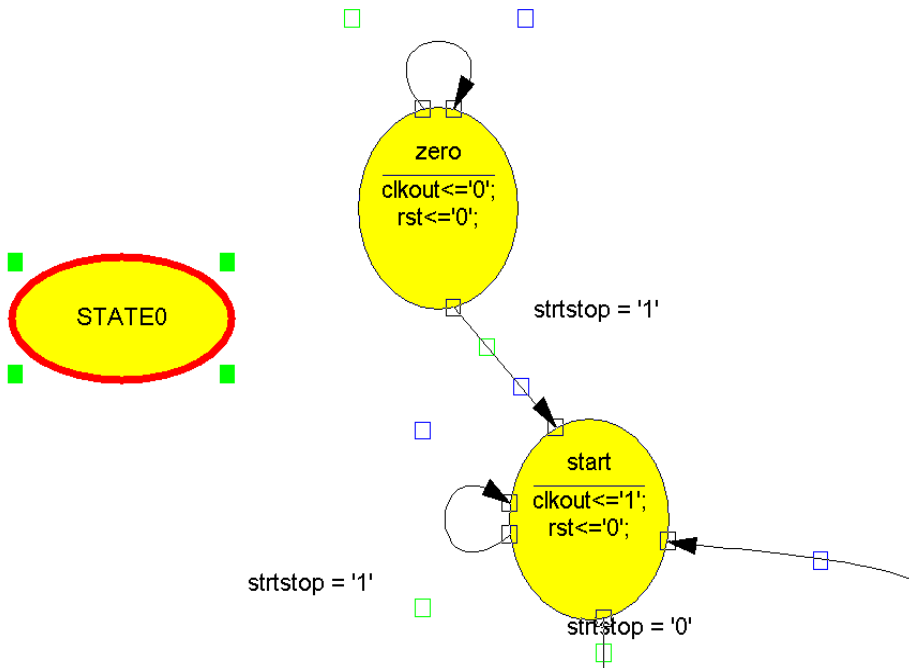


Figure 2-18 Adding the CLEAR State

To change the shape of the state bubble, click the bubble and drag it in the direction you wish to “stretch” the bubble.

Adding a Transition

A transition defines the movement between states of the state machine. Transitions are represented by arrows in the editor. You will be adding a transition from the **CLEAR** state to the **ZERO** state in the following steps. Because this transition is unconditional, there is no Transition Condition associated with it.

1. Click the Draw Transitions icon in the vertical toolbar.



2. Click twice on the **clear** state, once to select it, then once to start the transition.
3. Click the **zero** state to complete the transition arrow.
4. To manipulated the arrow's shape, click it and drag the mouse.

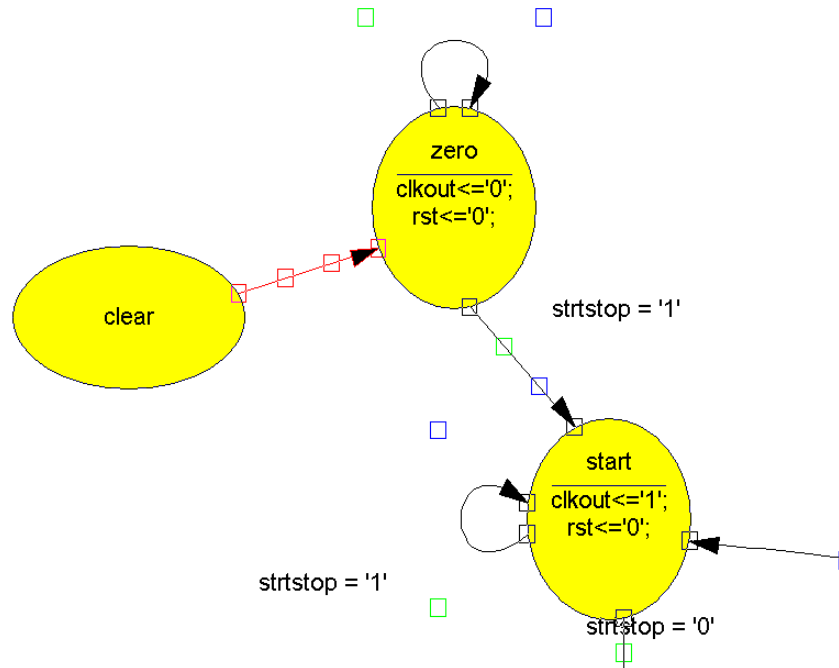


Figure 2-19 Adding State Transition

5. Click the Select Objects icon in the vertical toolbar.

Adding a State Action

A State Action dictates how the outputs should behave in a given state. You will add two state actions to the CLEAR state, one to drive the CLKOUT output to 0, and one to drive the RST output to 1.

To add a State Action:

1. Double-click the State.

The Edit State dialog box opens and you can begin to create the desired outputs.

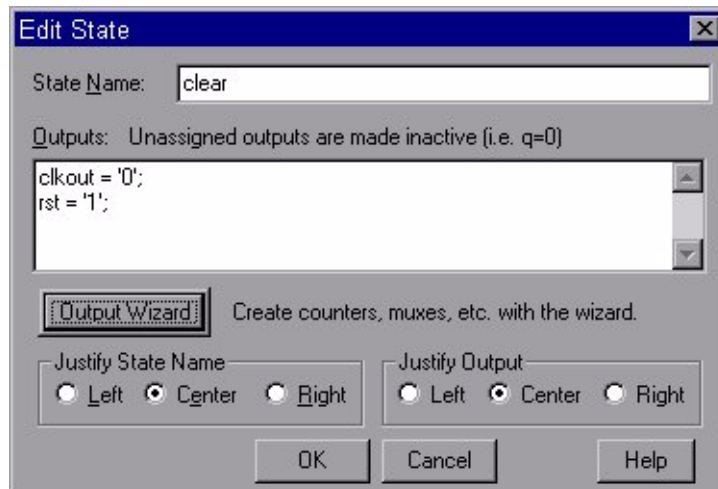


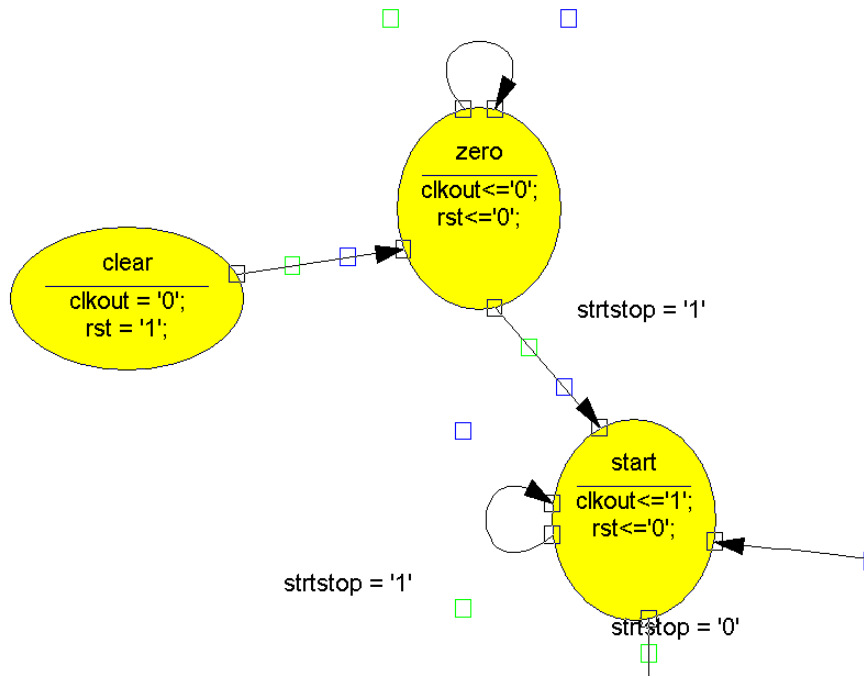
Figure 2-20 Edit State Dialog Box

2. Select the Output Wizard.
3. In the Output Wizard, select the following values:

```
DOUT = clkout, CONSTANT = '0';
```

```
DOUT = rst, CONSTANT = '1';
```

Click **OK** to enter each individual value.
4. Click **OK** to exit the Edit State dialog box. The outputs are now added to the state.



- Click the diagram near the **CLEAR** state, as shown in the diagram below.
- The cursor is automatically attached to the transition arrow for this Reset. Move the cursor to the **CLEAR** state, and click the state bubble.

4. A question is then asked, "Should this reset be asynchronous(Yes) or synchronous(No)?" Answer yes.

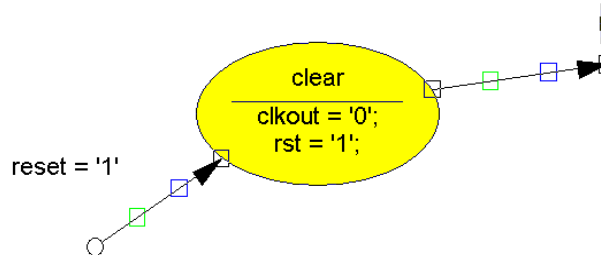


Figure 2-22 Adding Reset

5. Save your changes by selecting **File** → **Save**.

Creating the State Machine Macro

In this section, create the HDL code used to create a macro symbol that you can place on the Watch schematic. The macro symbol will automatically be added to the project library. Creation of the macro includes the creation of the HDL code from the state machine diagram.

1. Select **Options** → **Compile (Generate HDL)**.
State CAD verifies the state machine and displays the results.
2. Review the results and close the dialog.
State CAD will then create the HDL code and open a browser displaying the code.
3. Close the browser when you have finished examining the code.
4. Close State CAD.
5. In Project Navigator, select **Project** → **Add Source**.

6. Select `stmach_v.vhd` which is the VHDL file generated by State CAD.
7. Click **Open**.
8. Select VHDL module as the source type.
9. Click **OK**.

The file `stmach_v.vhd` is added to the project.

10. Select `stmach_v.vhd`.
11. Double-click Create Schematic Symbol under the Design Entry Utilities in the Processes for Current Source window.

Placing the STMACH, Tenths, and decode symbols

You can now place the STMACH, Tenths, and decode symbols on the Watch schematic.

1. If it is not already opened, open the Schematic Editor.
2. View the list of available library components in the Symbol Browser window.

You should now be able to locate the macros in the Local Symbols library.

3. Select the appropriate symbol, and add it to the Watch schematic as shown below.

Note: Do not worry about drawing the wires to connect this symbol. You will connect the entire schematic later in the tutorial.

4. Save the schematic.

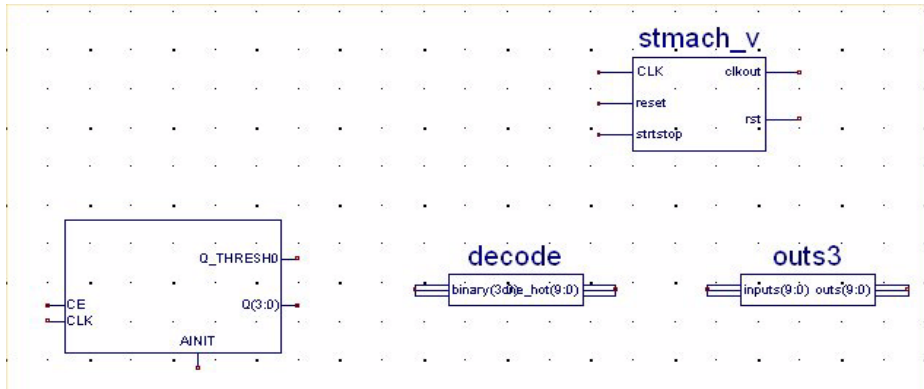


Figure 2-23 Placing the State Machine Macro

Creating an HDL-Based Module

With ISE, you can easily create modules from HDL code. The HDL code is connected to your top-level HDL design through instantiation and compiled with the rest of the design.

In this tutorial, create a new HDL module. This macro serves to convert the two 4-bit outputs of the CNT60 module into a 7-segment LED display format.

Using the HDL Design Wizard and HDL Editor

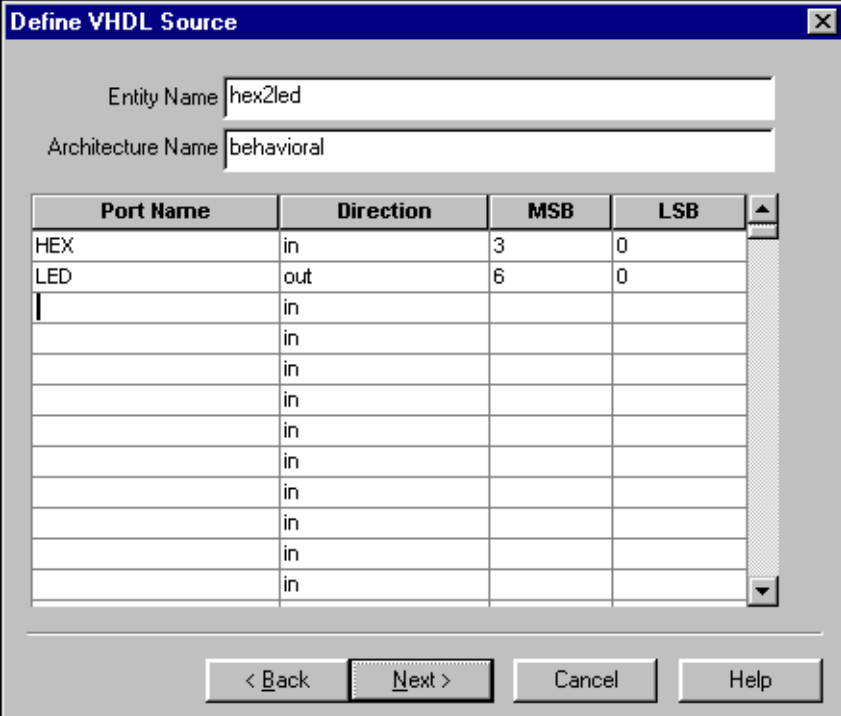
Enter the name and ports of the component in the HDL Wizard and the Wizard creates a “skeleton” HDL file which you can complete with the remainder of your code.

1. Select **Project** → **New Source**.
2. A dialog box opens, asking what type of source you want to create. Select VHDL or Verilog Module.
3. In the File Name field, type ‘hex2led’.
4. Click **Next**.

The hex2led component has a 4-bit input port named HEX and a 7-bit output port named LED.

5. To enter these ports, click in the Port Name field and type HEX.
6. Click in the Direction field and set the direction to in.

7. In the MSB field, enter 3, and in the LSB field, enter 0.



The image shows a 'Define VHDL Source' dialog box. It has two text input fields: 'Entity Name' with the value 'hex2led' and 'Architecture Name' with the value 'behavioral'. Below these is a table with four columns: 'Port Name', 'Direction', 'MSB', and 'LSB'. The table contains two rows of data: 'HEX' with direction 'in', MSB '3', and LSB '0'; and 'LED' with direction 'out', MSB '6', and LSB '0'. There are several empty rows below. At the bottom of the dialog are four buttons: '< Back', 'Next >', 'Cancel', and 'Help'.

Port Name	Direction	MSB	LSB
HEX	in	3	0
LED	out	6	0
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		
	in		

Figure 2-24 HDL Wizard

8. Repeat the previous steps for the LED(6:0) output bus. Be sure that the direction is set to out.
9. Click **Next** to complete the Wizard session.
A description of the module is now displayed.
10. Click **Finish** to open the “skeleton” HDL file in the HDL Editor.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity hex2led is
    Port ( LED : in std_logic_vector(3 downto 0);
          HEX : out std_logic_vector(6 downto 0));
end hex2led;

architecture behavioral of hex2led is

begin

end behavioral;
```

Figure 2-25 Skeleton VHDL File

```
module hex2led ( HEX,LED);
    input [3:0] HEX;
    output [6:0] LED;

endmodule
```

Figure 2-26 Skeleton Verilog File

In the HDL Editor, the ports are already declared in the HDL file, and some of the basic file structure is already in place. Keywords are printed in blue, data types in red, comments in green, and values are black. This color-coding enhances readability and recognition of typographical errors.

Using the Language Templates

The ISE language templates are HDL constructs and synthesis templates which represent commonly used logic components, such as counters, D flip-flops, multiplexers, and primitives. You can add your own templates to the Language Templates for components or constructs you use often.

To invoke the Language Assistant and select the template for this tutorial:

1. Select **Edit**→**Language Templates**.

Each HDL language in the Language Template is divided into four sections: Component Instantiations, Language Templates, Synthesis Templates, and User Templates. To expand the view of any of these sections, click the + next to the topic. Click any of the listed templates to view the template in the right hand pane.

2. Use the template called HEX2LED Converter located under the Synthesis Templates heading for VHDL or Verilog. Use the appropriate template for the language you are using.
3. To preview the HEX2LED Converter template, click the template in the hierarchy and the contents are displayed in the right-hand pane.

This template provides source code to convert a 4-bit value to 7-segment LED display format.

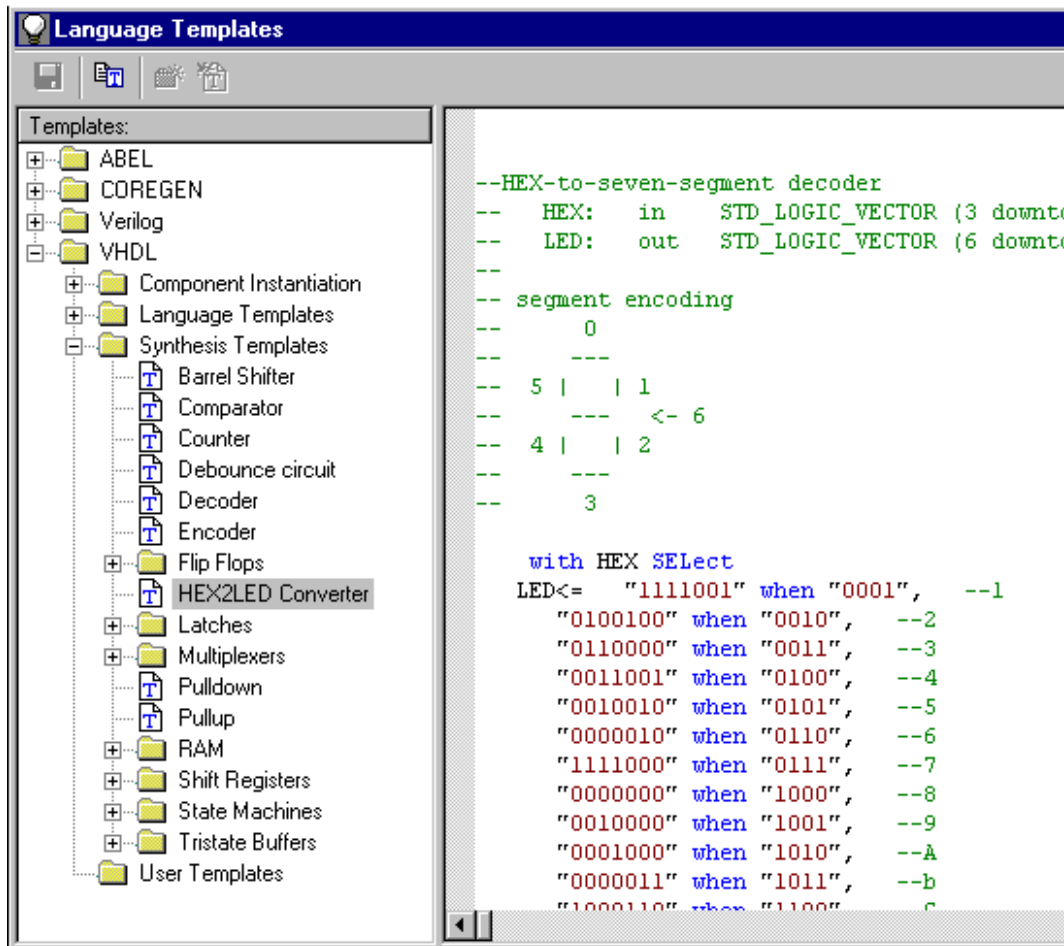


Figure 2-27 Language Templates

There are three ways of adding templates to your HDL file:

- Cut and paste contents directly from the Language Template.
- Drag and drop from the Language Template.
- The right-click menu, click Copy, then Paste.

The tutorial describes the drag and drop method.

4. In the Language Template, click and drag the HEX2LED Converter name into the hex2led.vhd under the architecture begin statement or the hex2led.v file under the module declaration.
5. Close the Language Assistant by clicking the X in the upper-right corner of the window.
6. (Verilog only) After the input and output statements and before the HEX2LED converter that you just added, add the following line of code to the HDL file to allow an assignment.

```
reg LED;
```

You now have complete and functional HDL code.

7. Save the file by selecting **File** → **Save**.
8. Select hex2led in the Sources in Project window and double-click the Check Syntax under Synthesize in the Processes for Current Source window.
9. Exit the HDL Editor.

Creating the HEX2LED symbol

The symbol representing the HEX2LED VHDL is created in Project Navigator.

1. In the Sources in Project window, select hex2led.vhd.
2. In the Processes for Current Source window, click the + beside Design Entry Utilities to expand the hierarchy.
3. Double-click Create Schematic Symbol.

Adding the HEX2LED Component to the Schematic

You are now ready to place the HEX2LED macro on the Watch schematic.

1. Open the Schematic Editor if it is not already open.
2. Open the Symbols Libraries dialog (refer to the [“Adding Components to CNT60”](#) section) to view the list of available library components.

You should now be able to locate the HEX2LED macro in this list.

3. Select HEX2LED, and add it to the Watch schematic, as shown in [Figure 2-28](#).

This component will be placed on the Watch schematic sheet in two separate instances.

To duplicate the component in the schematic:

1. Click the left mouse button while the pointer is on the placed symbol.
2. Click again to place the duplicate symbol.

Note: The Symbol Libraries icon must still be depressed on the Tools toolbar to enable this feature to automatically duplicate a symbol.

Again, do not worry about drawing the wires and buses to connect this macro. You will connect the entire schematic later in the tutorial.

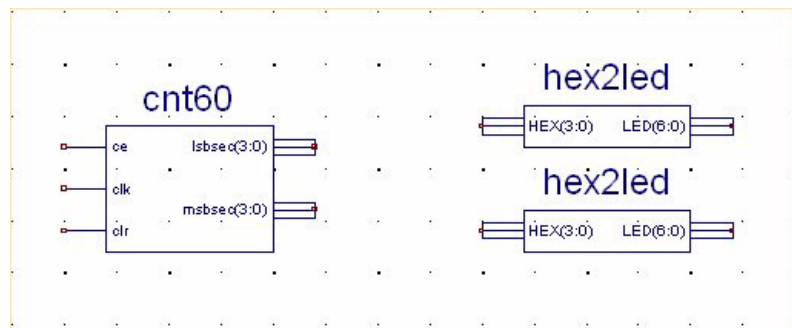


Figure 2-28 Placing the HEX2LED Component

Specifying Device Inputs/Outputs

When specifying device I/O on a schematic sheet, use the I/O Marker. All ECS schematics are netlisted to VHDL or Verilog and then synthesized by the synthesis tool of choice. When the synthesis tool synthesizes the top level HDL, the I/O markers are replaced with the appropriate pads and buffers.

Note: I/O components are contained in the Xilinx Unified Library and can be added to the schematic sheet; however, special considerations must be made.

- I/O macros are not recognized by the synthesis tool, therefore a duplicate buffer will be inserted. To avoid this use I/O primitives only.
- Global buffers, for example BUFG, are not recognized by the implementation tools, therefore a pad is not inserted and the logic is trimmed. To avoid this place a IBUF in front of the BUFG.

Hierarchy Push/Pop

Descend into a lower-level of hierarchy to view the underlying file. You will be pushing down into the OUTS3 macro, which is a schematic-based user-created macro.

To push down into OUTS3:

1. Select the symbol, then click the Hierarchy Push/Pop icon or right-click the macro and click Push into Symbol.



Figure 2-29 Hierarchy Push/Pop Icon

In the OUTS3 schematic, you see a series of inverters (INV) and output buffers (OBUF). This macro illustrates that you can place I/O buffers in a lower level macro. The output buffers are not required because the synthesis tool will insert buffers if one is not found.

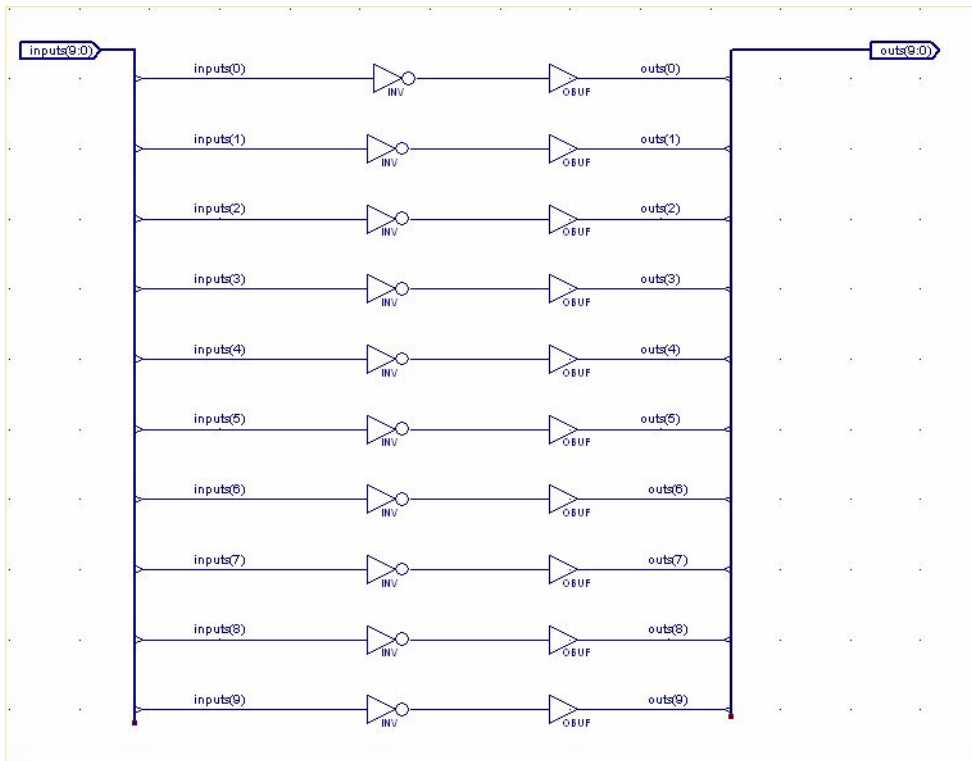


Figure 2-30 OUTS3 Schematic Macro

2. “Pop” back out of the OUTS3 component by clicking the Hierarchy Push/Pop icon.

Adding Input Pins

Add three more input pins to the Watch schematic, called CLK, RESET and STRTSTOP.

Add an IBUF for each of the two input pins, RESET and STRTSTOP. Add an IBUF and a BUFG for the input clock signal, CLK. To add these components:

1. Click the Add Symbol icon in the toolbar to open the Symbol Browser dialog.
2. Browse to locate the IBUF, INV, and BUFG components in the library.
3. Drop these on the schematic as shown below.
4. Draw a net between the IBUF, INV, and BUFG inputs and outputs. If necessary, refer to the section on drawing nets (see the [“Drawing Wires”](#) section) for instructions.
5. Draw a net to the input of each IBUF.

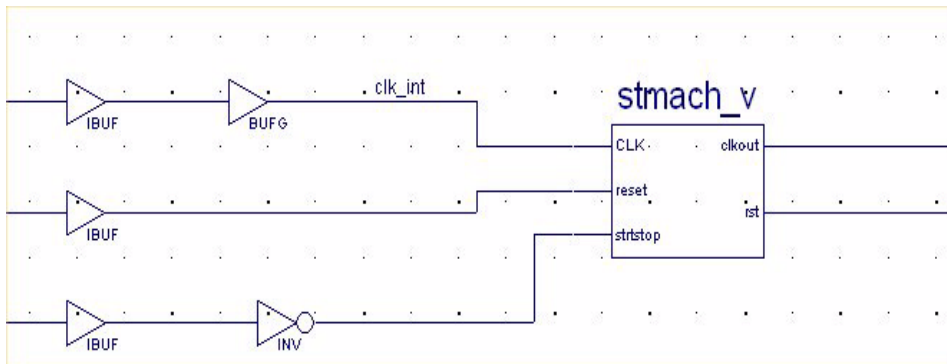


Figure 2-31 Placing CLK, RESET and STRTSTOP I/O Components

Adding I/O Markers and Net Names

It is important to label nets and buses for several reasons.

- It aids in debugging and simulation, as you will more easily trace nets back to your original design.
- Any nets which remain unnamed in the design will be given machine-generated names which will mean nothing to you later in the implementation process.
- Naming nets also enhances readability and aids in documenting your design.

Label the three input nets you just drew as illustrated in the completed schematic.

1. Select **Add → Net Name**.

The options toolbar changes to the following:



Figure 2-32 Options Toolbar for Add Net Name

2. Type the name of the net into the empty box.
The net name is now attached to the cursor.
3. Place the name on the leftmost end of the net as illustrated in [Figure 2-33](#).
4. Repeat Steps 1 through 3 for the STRTSTOP and CLK pins.

Once all of the nets have been labeled, add the I/O marker.

5. Select **Add → I/O Marker**.
6. In the I/O marker Options toolbar, select input then draw a box around the three labeled nets.

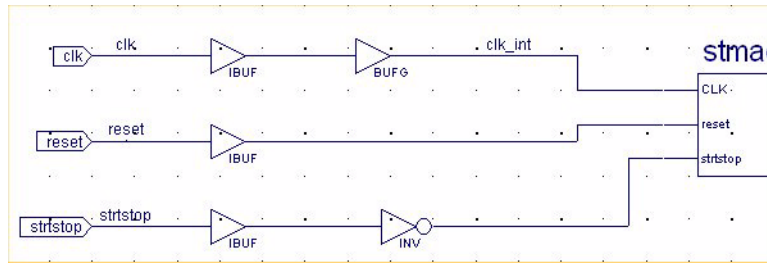


Figure 2-33 Labeled Nets

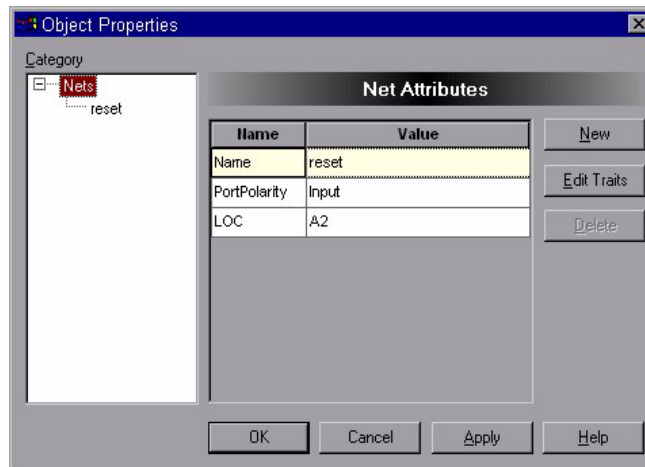
Assigning Pin Locations

Xilinx recommends that you let the automatic placement and routing program, PAR, define the pinout of your design. Pre-assigning locations to the pins can sometimes degrade the performance of the place and route tools. However, it is usually necessary, at some point, to lock the pinout of a design so that it can be integrated into a PCB (Printed Circuit Board).

Define the initial pinout by running the place-and-route tools without pin assignments, then locking down the pin placement so that it reflects the locations chosen by the tools. Because this design is simple and timing is not critical, the example pin assignments will not adversely affect the ability of PAR to place and route the design.

Specify pin locations by attaching a LOC parameter to a buffer component or I/O net. Assign a LOC parameter to the RESET net on the Watch schematic as follows.

1. Right-click either the RESET I/O marker, or the net connected to it, then select Object Properties.
2. Click the **New** button in under Net Attributes to add a new property.
3. Enter LOC for the Attribute Name and A2 for the Attribute Value.
4. Click **OK** to return to the Object Properties dialog box, and click **OK** once more to return to the schematic.

**Figure 2-34 Assigning Pin Locations**

To view the location constraint on the net add a net attribute window.

5. Select **Add** → **Attribute Window**.
6. Type LOC in the Attribute Name dialog.
7. Move the cursor to the RESET net connected to the I/O marker, and click anywhere on the net.
8. Repeat steps 1 through 4 to assign the STRTSTOP input to pin A5 and the CLK input to pin A3.

Completing the Schematic

Complete the schematic by wiring the components you have created and placed, adding any additional necessary logic, and labeling nets appropriately. The following steps guide you through the process of completing the schematic, or you may want to use the completed schematic shown below for guidance. Each of the actions in this section has been discussed in detail in earlier sections of the tutorial. If you need to review these sections, you may return to them. The finished schematic is shown in the following figure as a guide.

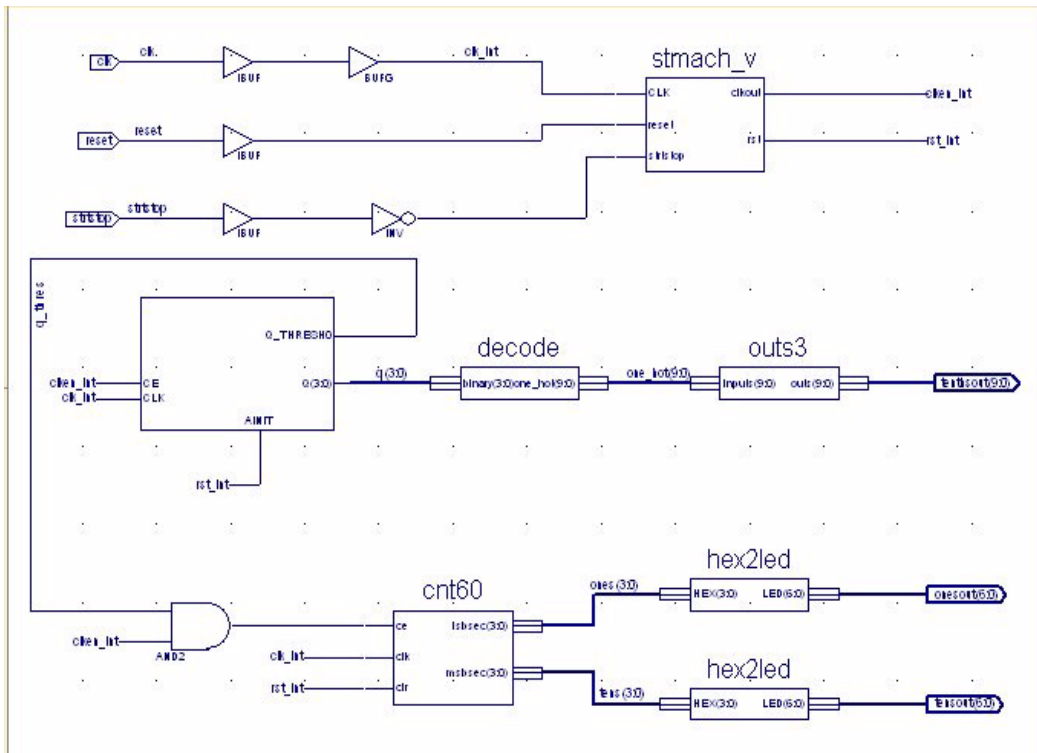


Figure 2-35 Completed Watch Schematic

1. Draw a net (see the [“Drawing Wires”](#) section) between the BUFG and the CLK pin of the STMACH state machine macro.
2. Label this net CLK_INT.

3. Draw a net (see the [“Drawing Wires” section](#)) between the IBUF of the RESET input and the RESET pin of the STMACH state machine macro.
4. Place an INV (inverter) component (see the [“Adding Components to CNT60” section](#)) from the Virtex library between the IBUF of the STRTSTOP input and the STRTSTOP pin of the STMACH state machine macro.
5. Draw nets (see the [“Drawing Wires” section](#)) to connect the INV to both the IBUF and the STMACH state machine macro.
6. Place an AND2 component (see the [“Adding Components to CNT60” section](#)) to the left of the CNT60 macro.
7. Draw a net (see the [“Drawing Wires” section](#)) to connect the output of the AND2 with the CE pin of the CNT60 macro.
8. Draw a net (see the [“Drawing Wires” section](#)) to connect the Q_THRES0 pin of the TENTHS macro to one of the inputs to the AND2.
9. Draw a hanging net (see the [“Drawing Wires” section](#)) from the CLKOUT pin of the STMACH macro. To terminate a hanging wire, double-click.
10. Name the new added net CLKEN_INT.
11. Draw a hanging net at the CLK_EN input pin of the TENTHS macro. Label this net CLKEN_INT (see the [“Adding I/O Markers and Net Names” section](#)).
12. Draw a hanging net (see the [“Drawing Wires” section](#)) at the other input of the AND2 component. Label this net CLKEN_INT again (see the [“Adding I/O Markers and Net Names” section](#)).
13. Draw a hanging net (see the [“Drawing Wires” section](#)) from the RST output pin of the STMACH macro.
14. Label this net RST_INT.
15. Draw two more hanging nets (see the [“Drawing Wires” section](#)), also named RST_INT, from the AINIT pin of the TENTHS macro and from the CLR pin of the CNT60 macro.
16. Draw two hanging nets (see the [“Drawing Wires” section](#)), each named CLK_INT, from the CLOCK pin of the TENTHS macro and from the CLK pin of the CNT60 macro.

Note: Remember that nets are logically connected if their names are the same, even if the net is not physically drawn as a connection in the schematic. This method is used to make the logical connection of the RST_INT, CLKEN_INT and CLK_INT signals.

17. Draw buses (see the [“Adding Buses”](#) section) to complete the schematic. Label them as shown on the preceding schematic diagram.

The schematic is now complete!

18. Save the design by selecting **File** → **Save**.

Behavioral Simulation

You can perform behavioral (also called functional) simulation before design implementation to verify that the logic that you have created is correct. Xilinx ISE provides integration with any ModelSim Simulator (ISE produces generic HDL netlists that work with most HDL simulators; however, the integrated flow is only available with MTI ModelSim). You can perform behavioral simulation on a schematic-based or HDL-based design. In a later section, you can perform timing simulation, which takes place after the design is implemented (placed and routed) with the Xilinx Implementation Tools.

This chapter contains the following sections.

- [“Overview of Behavioral Simulation Flow”](#)
- [“Adding an HDL Testbench”](#)
- [“Creating a Testbench Waveform using HDL Benchner”](#)
- [“Behavioral Simulation using ModelSim”](#)

Overview of Behavioral Simulation Flow

Behavioral simulation is an integral part of any HDL design flow as it enables a logical (functional) check of the design before any additional time is invested in Synthesis and Implementation. Xilinx ISE 4 provides a tightly integrated functional simulation flow with any version of ModelSim (release 5.4 and newer).

Using ISE, behavioral simulation can be conducted using either a hand-written HDL testbench (PC and Unix), or one generated automatically by Xilinx HDL Benchner (PC Only).

Note: Since Xilinx HDL Benchner is only available on the PC platform, Unix machine users should skip the automated testbench generation section which uses HDL Benchner. PC users can choose either flow.

Required Files

The funtional simulation flow requires the following files:

- Design files (VHDL, Verilog, or Schematic). These are produced by the designer, or from a HDL generation tool such as Xilinx StateCAD.
- Stimulus file, known as the testbench (VHDL, Verilog). Testbenches can be hand-written or produced using Xilinx HDL Benchner. To produce a testbench using HDL Benchner, refer to the HDL Benchner section in this chapter. The HDL Benchner flow is only available on the Windows platforms.
- ModelSim Script file to run the simulation (optional). Alternatively, the commands can be entered one-by-one into the simulator. Xilinx ISE creates the script file needed to run simulation in ModelSim.
- Xilinx Simulation Libraries, if any Xilinx primitive is instantiated in the design. More details on the libraries and how to compile them is provided in the next section.

Xilinx Simulation Libraries

To simulate designs containing Xilinx primitives with ModelSim in VHDL (or Verilog), you need the following simulation libraries which you must compile.

Unisims Library

The Unisim library is used for behavioral (RTL) simulation with instantiated components in the netlist, and for post-synthesis simulation. The recommended mapping name for the VHDL Unisims library is UNISIM and for the Verilog Unisims library is UNISIMS_VER. Additionally, there is a separate Unisims library in Verilog for simulating CPLD designs. This library is called UNI9000 and the recommended mapping name for this library is UNI9000.

LogiBLOX Library (VHDL Only)

The LogiBLOX library is used for designs containing LogiBLOX components, during pre-synthesis (RTL), and post-synthesis simulation. Since LogiBLOX models are not supported in Virtex, this library will not be used in this tutorial.

XilinxCoreLib Library

The XilinxCoreLib library must be used if a CoreGEN component is instantiated in the design. The recommended mapping name for the VHDL library is XILINXCORELIB where as the recommended mapping name for Verilog is XILINXCORELIB_VER.

All VHDL simulation libraries are provided at \$XILINX/vhdl/src and all Verilog simulation libraries are provided at \$XILINX/verilog/src.

For detailed instructions on compiling these libraries, see Xilinx Solution # 2561, which can be accessed as follows:

1. Go to <http://support.xilinx.com>.
2. Enter '2561' in the search box, and check to see that the search engine is pointing to Answer Records.
3. Click **OK**.

Solution 2561 will be displayed on the next page.

4. Click Solution 2561 on the next page.

Before compiling the libraries, notice that there is a file called modelsim.ini in the ModelSIM install directory. View this file and notice that the upper portion defines the locations of the compiled libraries. When doing a simulation, the modelsim.ini file must be provided either by copying the file directly to the directory where the HDL files are to be compiled and the simulation is to be run, or by setting the MODELSIM environment variable to the location of your master .ini file. You must set this variable since the ModelSim installation does not initially declare the path for you.

For UNIX, type the following environment variable:

```
setenv MODELSIM /path/to/the/modelsim.ini
```

For PCs, set the MODELSIM environment variable to path where the modelsim.ini file is located in a similar fashion. To set the environment variable, go to **Start** → **Settings** → **Control Panel** → **System** → **Environment**.

Adding an HDL Testbench

This section demonstrates how to add pre-existing testbench files to the project. This design flow is for users of UNIX machines who cannot generate testbenches using HDL Benchner. PC machine users can choose to use hand-written testbenches for their design.

The testbench must be associated with the top-level design file in order to successfully simulate the design.

VHDL design

For a VHDL design, add your testbench as follows:

1. Select **Project** → **Add Source**.
2. Select the testbench file stopwatch_tb.vhd and click **Open**.

The Choose Source Type dialog box opens.

3. Select VHDL Testbench and click **OK**.

ISE recognizes the top-level design file associated with the testbench and adds the testbench in the correct order

Verilog design

For a Verilog design, add your testbench as follows. Ensure the extension of the testbench file is .tf rather than .v.

1. Select **Project** → **Add Source**.
2. Select the testbench file stopwatch_tb.vhd and click **Open**.

ISE recognizes the top-level design file associated with the testbench and adds the testbench in the correct order.

Creating a Testbench Waveform using HDL Bench

HDL Bench is a PC-based testbench/test fixture creation tool that is part of ISE 4. This tool allows you to graphically enter stimulus and the expected response, then generates a VHDL testbench or Verilog test fixture.

Creating a Testbench Waveform Source

Use the following procedure to create a testbench/test fixture with HDL Bench.

1. Select stopwatch in the Sources in Project window.
2. Select **Project** → **New Source** from the Project Navigator menu.
3. In the New dialog box, select Test Bench Waveform as the source type.
4. Type the name 'stopwatch_tb'.
5. Select **Next**.

Note: In the Select dialog box, the stopwatch file is the default source file because it is selected in the Sources in Project window (step 1).

6. Select **Next** and select **Finish**.

HDL Bench is launched and you are prompted to specify the timing parameters used during simulation. The clock high time and clock low time together define the clock period for which the design must operate. The Input setup time defines when inputs must be valid. The Output valid delay defines the time after active clock edge when the outputs must be valid.

For this tutorial, the defaults are used.

7. Click **OK** to accept the default timing constraints.

HDL Bench now opens with two main windows. The top window is the Waveform window. In this window, enter graphically depictions of the stimulus and expected response. The bottom window is the currently loaded HDL file.

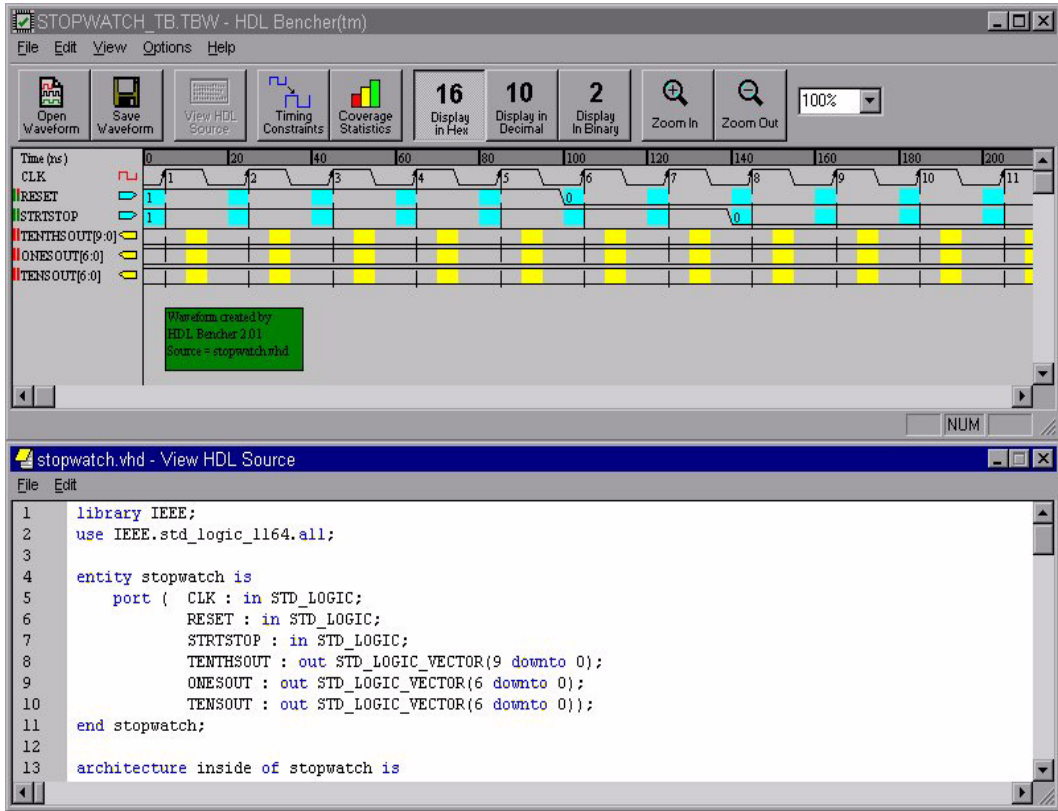


Figure 3-1 HDL Benchers Windows

Initializing Inputs

Enter the following input stimulus:

1. Click the RESET cell under CLK cycle 1 to set it high.
2. Click the RESET cell under CLK cycle 5 to set it low.
3. Click the STRTSTOP cell under CLK cycle 30 to set it high.
4. Click the STRTSTOP cell under CLK cycle 33 to set it low.
5. Click the STRTSTOP cell under CLK cycle 39 to set it high.
6. Click the STRTSTOP cell under CLK cycle 41 to set it low.
7. Click the Save Waveform icon in the toolbar.

Next, HDL Benchner will prompt you to set the number of clock cycles for which you wish to simulate.

8. Enter 9 in the dialog box.

This extends the waveform 9 clock cycles past the assertion of STRTSTOP low, for a total of 50 clock cycles for the testbench.

9. Click **OK**.

10. Exit HDL Benchner

The new testbench waveform source (stopwatch_tb.tbw) is automatically added to the project. In future, you can open HDL Benchner from Project Navigator by double-clicking on this file.

Generating Expected Results

Using HDL Benchner, you can generate expected outputs for the stopwatch module based on the initialized inputs you entered.

1. Select stopwatch_tb in the Sources in Project window.
2. In the Processes for Current Sources window, click the + beside ModelSim Simulator to expand hierarchy.
3. Double-click Generate Expected Simulation Results.

This process invokes ModelSim in the background and runs a simulation using the inputs specified, generating output values which are added to the testbench waveform.

4. In HDL Benchner, click on the Display in Binary icon in the toolbar menu if it is not already selected. By doing this you can view your results in binary rather than hexadecimal or decimal.
5. Using the scroll-bars and Zoom In/Out toolbar icons, compare your new waveform to the one in [Figure 3-2](#).
6. Exit HDL Benchner without saving the waveform.

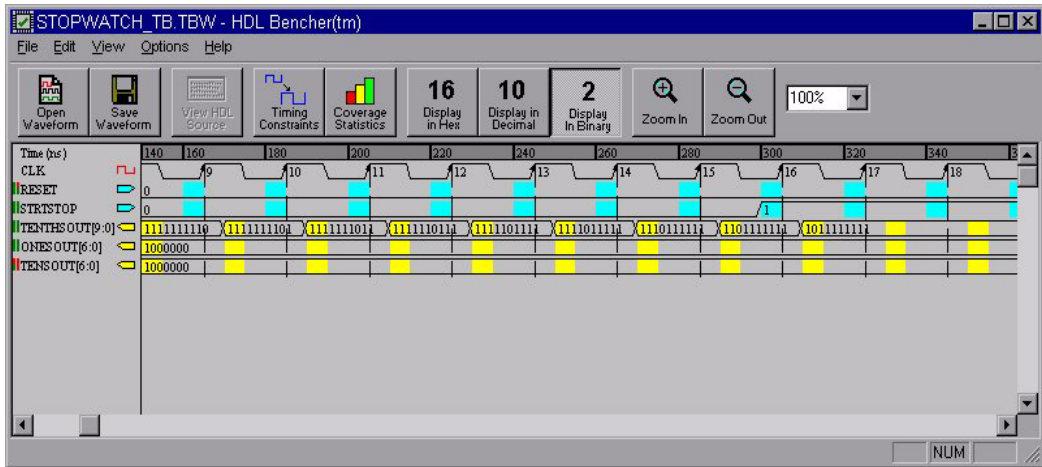


Figure 3-2 Expected Response

Behavioral Simulation using ModelSim

ISE has full integration with any version of the ModelSim Simulator. ISE provides work directory creation, source file compilation, simulation initialization, and control over simulation properties.

Selecting Simulation Processes

Four simulation processes are available and are briefly defined in this section.

To display the ModelSim Simulator Processes:

1. In the Sources in Project window, select stopwatch_tb.vhd (stopwatch_tb.tf for Verilog).
2. Click the + beside ModelSim Simulator to expand this process hierarchy.

The following simulation processes are available:

- Simulate Behavioral VHDL (or Verilog) Model — This is the process we will kick off to simulate our RTL design.
- Simulate Post-Translate VHDL (or Verilog) Model — This will simulate the netlist after the NGDBUILD implementation stage.
- Simulate Post-Map VHDL (or Verilog) Model — This will simulate the netlist after the Map implementation stage.
- Simulate Post-Place & Route VHDL (or Verilog) Model — This will simulate the back-annotated netlist after Place & Route, which contains the detailed timing information as well.

In this tutorial, you will perform a behavioral simulation on the stopwatch design. But first, you must specify the process properties for simulation.

Specifiying Simulation Properties

ISE allows you to set several MTI Simulator properties in addition to the simulation netlist properties. In order to see which properties are available for RTL simulation, step through the following command sequence in ISE:

1. In the Sources in Project window, select `stopwatch_tb.vhd` (`stopwatch_tb.tf` for Verilog).
2. Click the + sign next to ModelSim Simulator in the Processes For Current Source window
3. Right-click on Simulate Behavioral VHDL (or Verilog) Model.
4. Select Properties.

The Process Properties dialog box (Figure 3-3) contains the following simulation properties which can be specified or changed as indicated below:

- Custom Do File — This option allows a different .do run script to be specified.
- Use Automatic Do File — When unchecked, this option will bring up ModelSim but not automatically run the processes required to simulate the design. You will have to manually run the .do file from ModelSim or enter the commands one-by-one to run simulation.
- Simulation Run Time — Specifies default time for which simulation is run.
- Simulation Resolution — This is set to 1 ps by default, but can be changed as required.
- Design Unit Name — This property allows you to specify the top-level model to be loaded in ModelSim. This property must be changed if the top level entity, configuration, or module is named something different than testbench.

The Second tab in this GUI is for selecting Display Properties. Using this tab, you can select which ModelSim Simulation windows will automatically be invoked. By default, the Signals, Structure and Wave windows are invoked. For more details on ModelSim Simulator windows, refer to the *ModelSim User Manual*.

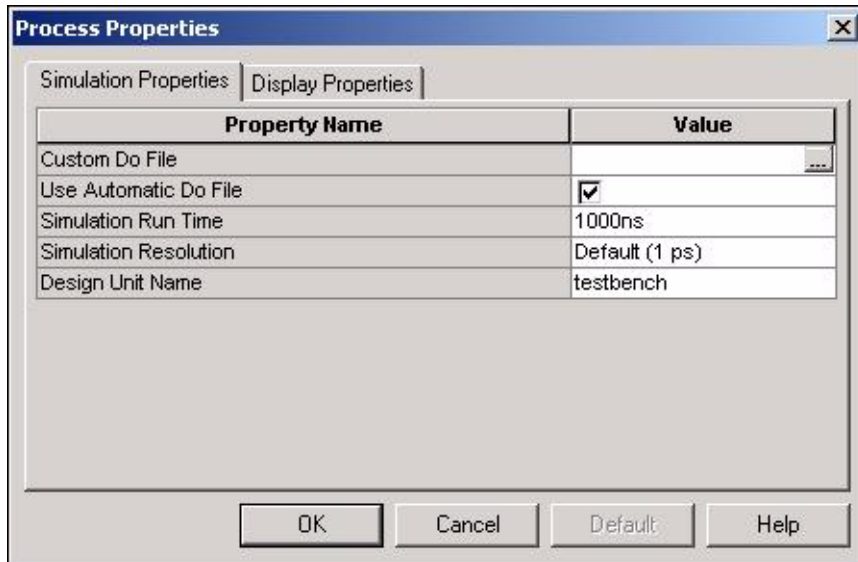


Figure 3-3 Process Properties Dialog Box

Performing Simulation

Once the process properties have been set, you are ready to run ModelSim.

To start the behavioral simulation:

1. Double-click Simulate Behavioral VHDL (or Verilog) Model under the ModelSim Simulation process.

ModelSim creates the work directory, compiles the source files, loads the design and performs simulation for the time specified.

There are two basic steps to simulate your design:

- [“Adding Signals”](#)
- [“Saving the Simulation”](#)

There are several different ways to perform each of these steps. These methods are discussed briefly in the following sections.

Adding Signals

In order to view signals during the simulation, you must first add them to the Wave window. ISE automatically adds all of the top-level ports to the Wave window. Additional signals are displayed in the Signal window based upon the selected structure in the Structure window.

There are two basic methods for adding signals to the Simulator Wave window.

- Drag and drop from the Signal window.
- **View → Wave → Selected Signals** from the Signal window.

The following procedure explains how to add additional signals in the design hierarchy. For the purpose of example, add the `lsbsec` and `msbsec` signals in the `cnt60` macro. To do so:

1. In the Structure window, click the + next to `uut:stopwatch(schematic)`.

Note: *uut* represents the instance in the testbench, *stopwatch* is the component/entity name and *schematic* is the architecture name.

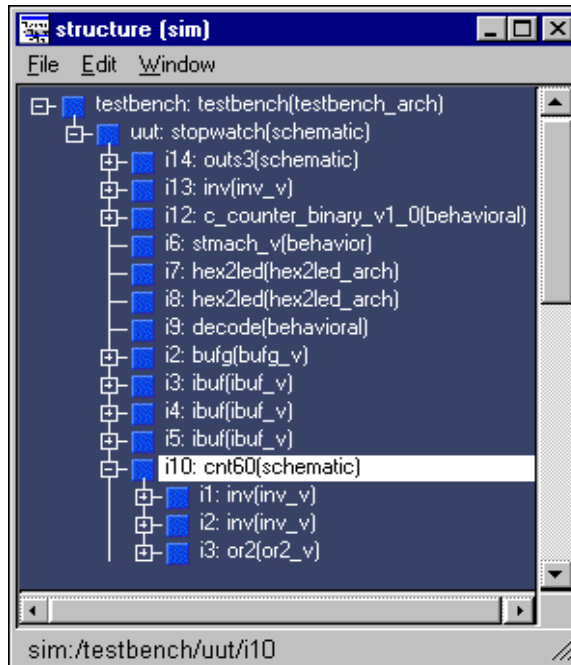


Figure 3-4 Structure Window

2. Select i10:cnt60(schematic) in the Structure window.

Notice that the signals listed in the Signal window are updated.

3. Click and drag lsbsec from the Signal window to the Wave window.
4. Select msbsec in the Signal window and select **View** → **Wave** → **Selected Signals** to add the signal to the Wave window.

Notice that the waveforms have not been drawn for lsbsec or msbsec.

There are two ways to add the waveforms for these signals:

- Continue with the simulation
- Restart the simulation.

To continue to run the simulation:

1. Click on the Continue Run icon on the toolbar.



Figure 3-5 Continue Run icon

To restart the simulation:

1. Click on the Restart Simulation icon.



Figure 3-6 Restart Icon

The Restart dialog box opens.

2. Click **Restart**.
3. (Optional) Select the Continue Run icon to re-run the simulation.

Saving the Simulation

The ModelSim Simulator provides the ability to save the signals list in the Wave window. This can be important when additional signals or stimulus have been added and the simulation must be restarted. The saved signals list can easily be loaded each time the simulation is started.

1. In the Wave window, select **File → Save Format**.
2. In the Save Format dialog box, change the default filename `wave.do` to `sec_signal.do`.

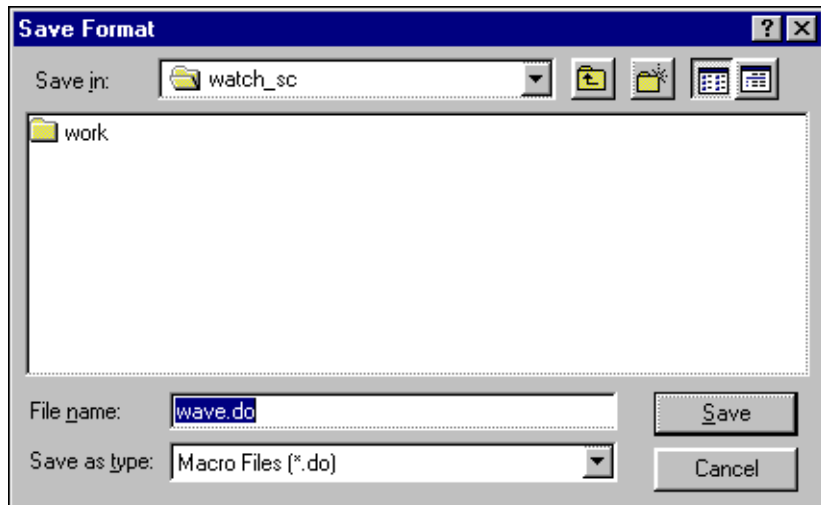


Figure 3-7 Save Format Dialog Box

3. Click **Save**.

In addition, the simulation results that appear in the waveform are also saved. These are saved in a file called `vsim.wlf`, which is produced by default in the ModelSim project directory (which is the same as the ISE project directory for ISE-MTI projects). If this file needs to be preserved, it must be copied or renamed, otherwise it will be overwritten upon restart of simulation.

To view the contents of this file later, type the following commands one-by-one at the ModelSim prompt:

```
Modelsim> vsim -view vsim.wlf
vsim> view wave
vsim> add wave *
```

The simulation output will then appear in the Wave window.

Restarting the Simulation

If you remember the original simulation resulted in errors between the expected and actual response of the simulation. These errors are the result of the reset signal being held high for too long. Close the simulator by selecting **File** → **Quit** in the ModelSim window.

These errors can easily be fixed by changing the stimulus in HDL Benchner. Open HDL Benchner by double-clicking on stopwatch_tb.tbw in the Source Window of Project Navigator. This will open HDL Benchner.

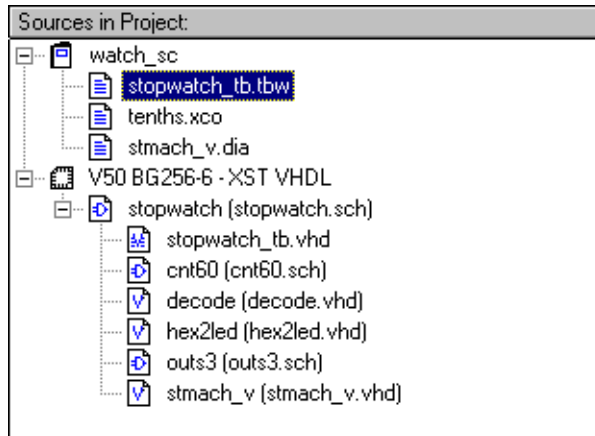


Figure 3-8 TBW File in Sources in Project Window

The following procedure explains how to make the changes and restart the simulation.

1. In HDL Benchner, click the reset cell under clk cycle 4 to set it low.
2. Save the waveform.
3. Close HDL Benchner.
4. To restart the simulation by double-clicking on the Simulate Behavioral VHDL (or Verilog) Model.
5. The Simulation will now complete without errors.

Design Implementation

Design Implementation is the process of translating, mapping, placing, routing, and generating a BIT file for your design. The Design Implementation tools are embedded into the ISE for easy access and project management.

This chapter is the first in the “[Implementation-only Flow](#)”, and an important chapter for the “[HDL Design Flow](#)” and the “[Schematic Design Flow](#).”

This chapter contains the following sections.

- “[Installing the Tutorial Files](#)”
- “[Creating an Implementation Project](#)”
- “[Specifying Options](#)”
- “[Translating the Design](#)”
- “[Using the Constraints Editor](#)”
- “[Mapping the Design](#)”
- “[Using the Floorplanner](#)”
- “[Using Timing Analysis to Evaluate Block Delays After Mapping](#)”
- “[Placing and Routing the Design](#)”
- “[Using FPGA Editor to Verify the Place and Route](#)”
- “[Evaluating Post-Layout Timing](#)”
- “[Creating Configuration Data](#)”
- “[Using the PROM File Formatter](#)”

Installing the Tutorial Files

This chapter demonstrates the ISE Implementation flow. The front-end design has already been compiled in an EDA interface tool and is described by an EDIF Netlist File (EDF). For a listing of EDA Interfaces, please reference the beginning of this tutorial. This tutorial passes an input netlist from the front-end tool to the back-end Design Implementation tools, and incorporates placement constraints through a User Constraints File (UCF). Timing constraints are added later through the Constraints Editor.

The tutorial design (Watch) is designed to perform like a track coach's stopwatch. There are two inputs to the system (RESET and SRTSTP). The configuration clock on the device is used as a ten-hertz clock signal. Three seven-bit outputs are generated by this system for output to three seven-segment LED displays. There are two options you can follow in this chapter's tutorial:

- Option 1 is to have gone through the previous chapters and synthesized the design to create the EDIF Netlist File. If you choose this option, please proceed to the [“Creating an Implementation Project”](#) section.
- Option 2 is to use the EDIF Netlist Files that are provided. If you choose this option, please create a working directory with the tutorial files as follows.
 - a) Create an empty working directory named Watch.
 - b) Copy the following files from <http://support.xilinx.com/support/techsup/tutorials/> directory into to your newly created working directory.

Table 4-1 Required Tutorial Files

File Name	Description
Stopwatch.edn or stopwatch.edf	Input netlist file (EDIF)
Tenths.edn	Counter netlist file (EDIF)
Watch.ucf	User Constraints File

Creating an Implementation Project

This section describes how to create a project through ISE. The process is the same for either Schematic or HDL designs.

To begin, use the following steps:

1. Perform one of the following:
 - On a workstation, enter the following to start ISE: **ise &**
 - On a PC, select the following to start ISE Project Navigator:
Start → Programs → Xilinx ISE 4 → Project Navigator.
2. Proceed as follows:
 - If you are continuing this project from the previous chapters, please proceed to the [“Specifying Options”](#) section.
 - If you are using the pre-synthesized design, create a new project and add the (stopwatch) EDIF netlist, as follows;
 - a) Select **File → New Project.**
 - b) Type “EDIF Flow” for the Project Name. Select Virtex2 for the Device Family, xc2v40-5fg256 for the Device, and EDIF for the Design Flow. Then select **OK.**

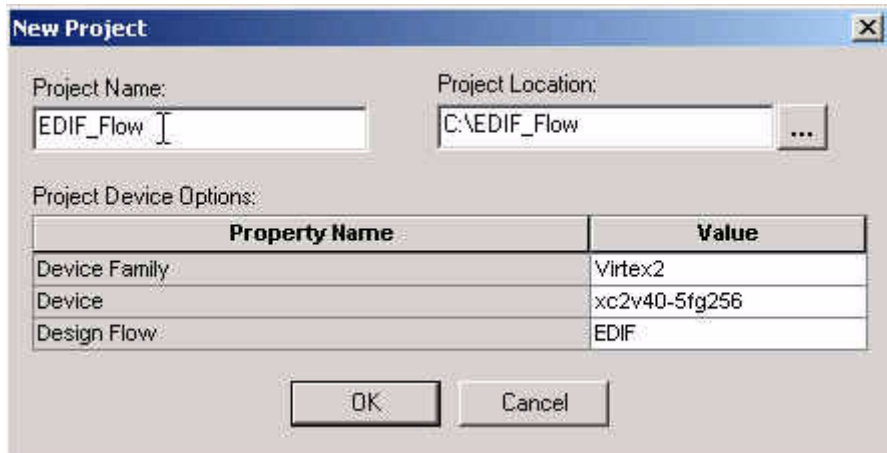


Figure 4-1 New Project Dialog Box

- c) After the project is created, then right-click xc2v40-5cs144.
- d) Select Add Sources and select stopwatch.edf or stopwatch.edn. Then select **Open**.

When you create a new project, you specify a design to open and a directory for the project. You can create as many projects as you want, but you can only work with one at a time.

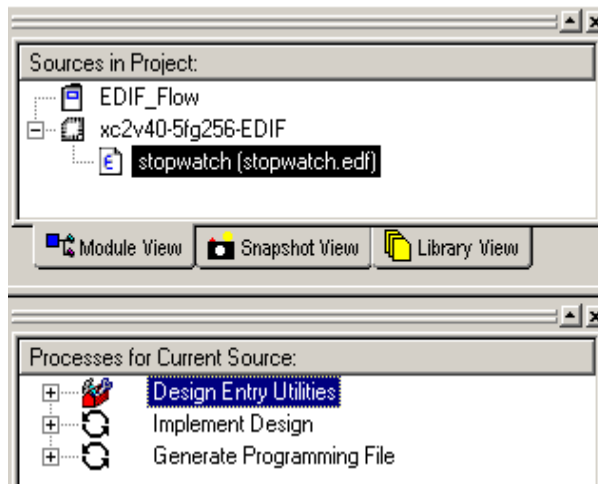


Figure 4-2 Selecting File in Sources in Project Window

- 3. In the Sources in Project window, select the top-level module, stopwatch (edf or edn). This enables the design to be implemented.

Specifying Options

In this section, you are going to set some options for implementation. In each dialog box, you can select the Help button to read more about the options and information regarding the dialog box. The implementation options control how the software maps, places, routes, and optimizes a design.

The implementation options for ISE is broken into two groups, Standard and Advanced. The default setting is Standard, which allows access to the most commonly used options. The Advanced settings provide access to all implementation options.

1. To enable the Advanced Options, select **Edit → Preferences**. In the Preferences dialog select the Processes Tab. Change the Property Display Level from Standard to Advanced. Select **OK**.

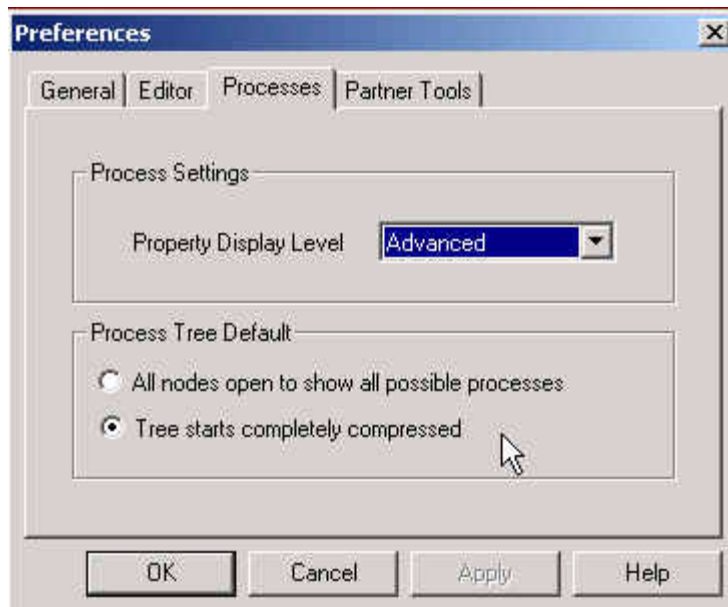


Figure 4-3 Preferences Dialog Box

2. To set more implementation options, right-click Implement Design, and select Properties. The Process Properties dialog provides access to Translate, Map, Place and Route, and Timing Report options.

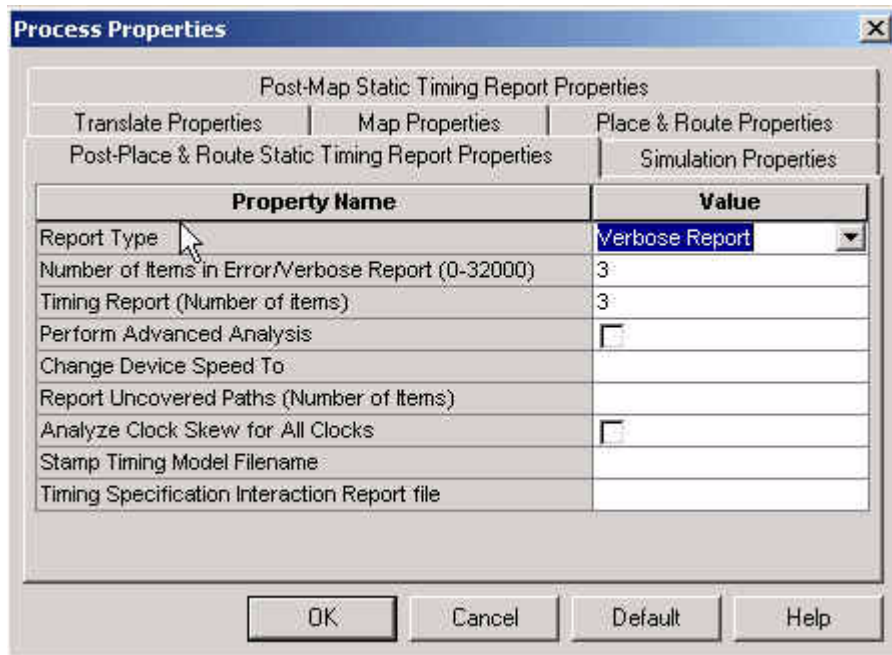


Figure 4-4 Post-Place & Route Static Timing Report Properties

3. In the Place & Route Properties tab, change the Place & Route effort level (overall) to Highest. This option increases the overall effort level of Place and Route during implementation.

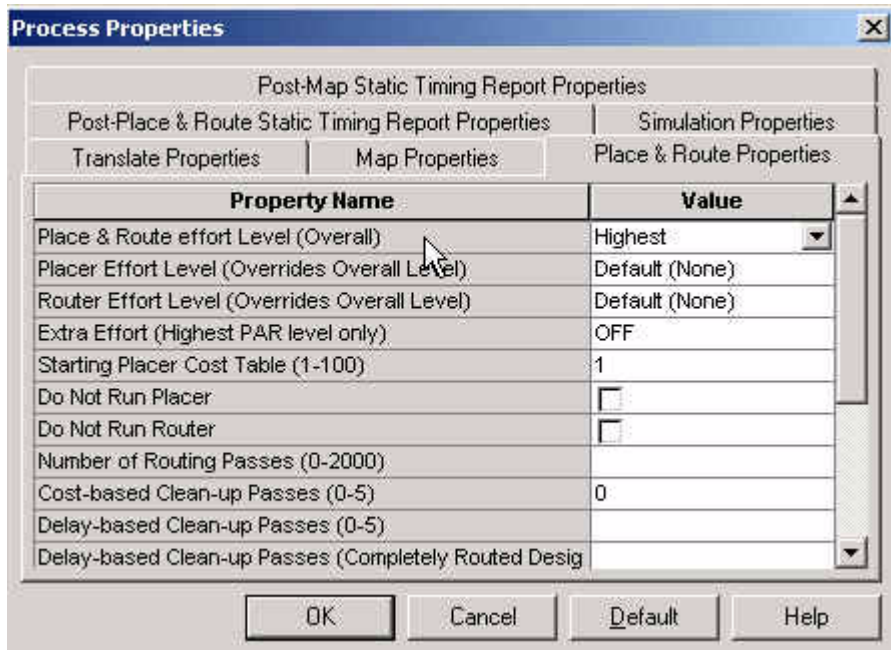


Figure 4-5 Place & Route Properties

4. In the Post-Place & Route Static Timing Report Properties tab, change Report type to Verbose Report.

This option changes the type of report from an error report to a verbose report. This report will be created after Place and Route is completed.

5. Select **OK** to exit the Process Properties dialog box.

The User Constraints File (UCF) provides a mechanism for constraining a logical design without returning to the design entry tools. However, it requires that you understand the exact syntax needed to define constraints. On the other hand, the Constraints Editor is a graphical tool in the Xilinx Development System that allows you to enter timing and pin location constraints.

6. In order to launch the Constraints Editor, expand the Design Entry Utilities tree, expand the User Constraints tree and double-click on Edit Implementation Constraints (Constraints Editor). This will automatically run the Translate step, which is discussed in the following section.

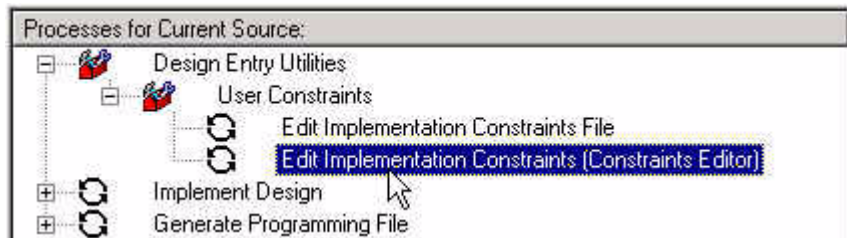


Figure 4-6 Edit Implementation Constraints

Translating the Design

ISE manages all of the files created during the implementation process. The programs run by ISE use the settings supplied by you in the options dialog box. This gives you complete control over how a design is processed. Typically, one sets all their options first and then runs through the entire flow by clicking Implement Design, and selecting Run. This tutorial is going to do the implementation one step at a time. During translation, the program NGDBuild is executed, and performs the following functions:

- Converts input design netlists and writes results to a single merged NGD netlist. The merged netlist describes the logic in the design as well as any location and timing constraints.
- Performs timing specification and logical design rule checks
- Adds the User Constraints File (UCF) to the merged netlist

Once these processes are all complete, ISE launches the Constraints Editor.

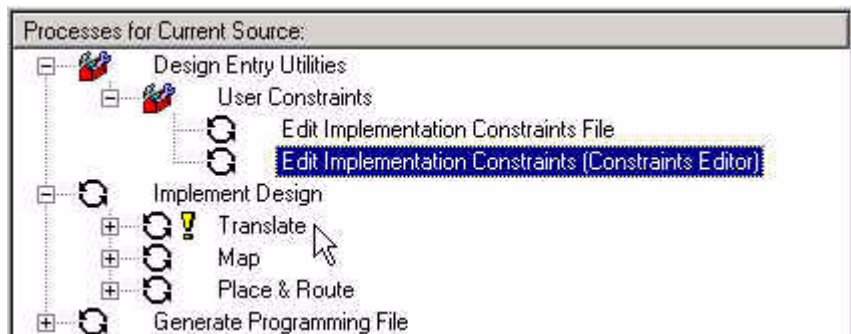


Figure 4-7 Launching Constraints Editor

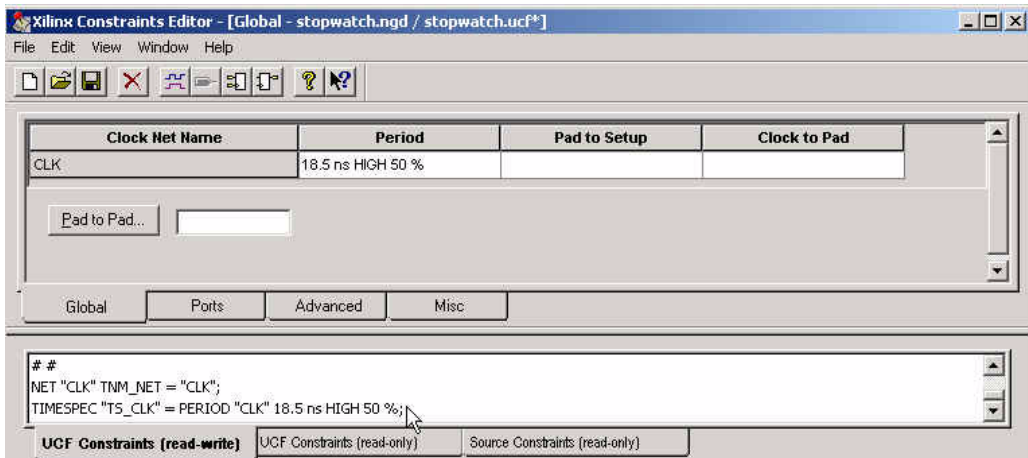
Using the Constraints Editor

The Constraints Editor is a utility that allows you to edit constraints previously defined (through a UCF file), as well as add new constraints to your design. Input files to the Constraints Editor are:

- NGD (Native Generic Database) file. This file serves as input to the mapper which then outputs the physical design database, an NCD (Native Circuit Description) file.
- Corresponding UCF (User Constraint File). By default, when the NGD file is opened, an existing UCF file with the same base name as the NGD file is used. Alternatively, you can specify the name of the UCF file.

The Constraints Editor generates a valid UCF file. The Translate step (NGDBuild) uses the UCF file, along with design source netlists to produce a newer NGD file, that incorporates the changes made. The MAP program (the next section in the design flow) then reads the NGD. In this design, the stopwatch.ngd file and stopwatch.ucf files are automatically read into the Constraints Editor.

The Global tab appears in the foreground of the Constraints Editor window. This window automatically displays all the clock nets in your design, and allows you to define the associated period, pad to setup, and/or clock to pad values.

**Figure 4-8 Constraints Editor**

Perform the following steps in the Constraints Editor:

1. Select the Period cell on the row associated with the clock net CLK.
2. Double-click your left mouse button. This invokes the Clock Period dialog box.
3. Within the Clock Signal Definition, keep the default (Specific Time) selected to define an explicit period for the clock rather designate a period which is relative to another timing specification.
4. Enter a value of 18.5 in the Time text box. Verify that ns is selected from the Units pull-down list. Click **OK**. Notice that the period cell is updated with the global clock period constraint that you just defined (with a default 50% duty cycle).

Note: For the purpose of this tutorial, you invoked a secondary dialog box by double-clicking a cell to specify your constraint values. Another feature is to do direct entry of constraints into cells by simply clicking once.

5. Select the Ports tab from the Constraints Editor's main window.

The left hand side displays a listing of all the current ports as defined by the user. Notice that certain cells in the Location column are pre-populated with device pins locking down ports to actual pins on the target device. This information was obtained by the Constraints Editor by way of the stopwatch.ucf file it read in.

6. Enter the pin locations by selecting the Location text box associated with each of the following signals:

```

onesout<0> -> H4
onesout<1> -> E3
onesout<2> -> E4
onesout<3> -> D2
onesout<4> -> D3
onesout<5> -> D1
onesout<6> -> C1

```

Port Name	Port Direction	Location
CLK	INPUT	
ONESOUT<0>	OUTPUT	H4
ONESOUT<1>	OUTPUT	E3
ONESOUT<2>	OUTPUT	E4
ONESOUT<3>	OUTPUT	D2
ONESOUT<4>	OUTPUT	D3
ONESOUT<5>	OUTPUT	D1
ONESOUT<6>	OUTPUT	C1

Figure 4-9 Constraint Editor's Port Tab

7. Select **File** → **Save**. The change made within the Constraints Editor is now saved into the stopwatch.ucf file in your current revision directory.
8. Select **File** → **Exit**. If you get a dialog box, asking to Reset the Implement Design process..., select Reset to reset the process.
9. Now, you will continue with the implementation, but translate will need to be re-run. Right-click Translate, and select Rerun.

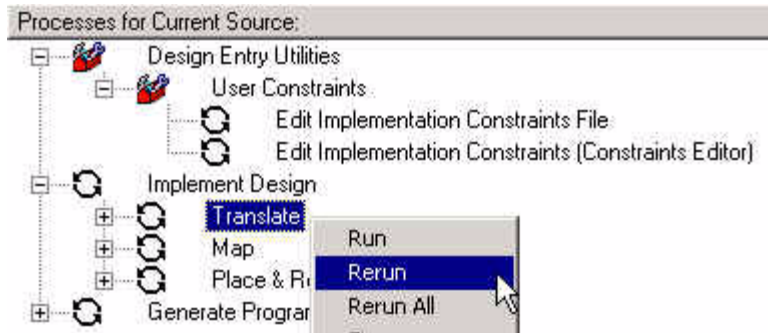


Figure 4-10 Rerun Translate Process

Mapping the Design

Now that all implementation strategies have been defined (options and constraints), continue with the implementation of our design.

1. Right-click Map.
2. Select Run in the pop-up menu.
3. Expand the Implement Design tree, to see the progress through implementation.

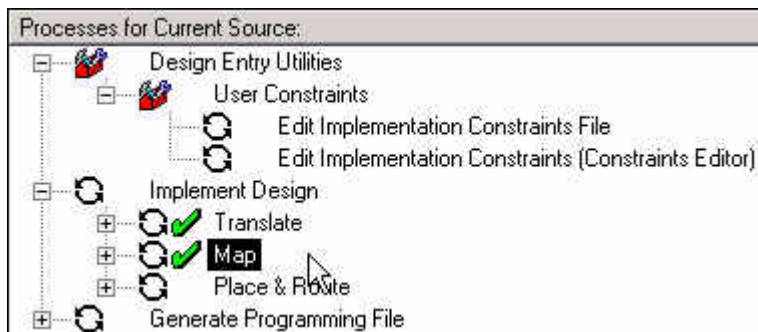


Figure 4-11 Mapping the Design

The design is being mapped into CLBs and IOBs. After mapping, the design will be placed and routed. The final step in the design flow is the Configure step in which a configuration bitstream is created for downloading to a target device or for formatting into a PROM programming file. Map performs the following functions:

- Allocates CLB and IOB resources for all basic logic elements in the design
- Processes all location and timing constraints, performs target device optimizations, and runs a design rule check on the resulting mapped netlist.

After each step is done, you will see that each step has generated its own report. Each report can be found by expanding either the Translate tree or the Map tree. The following reports are listed by their type and descriptions:

Table 4-2 Reports Generated Through MAP

Translation Report	Includes warning and error messages from the translation process.
Map Report	Includes information on how the target device resources are allocated, references to trimmed logic, and device utilization. For detailed information on the Map report, refer to the <i>Development System Reference Guide</i> .

4. Expand either the Translate tree or the Map tree and double-click one of the reports.

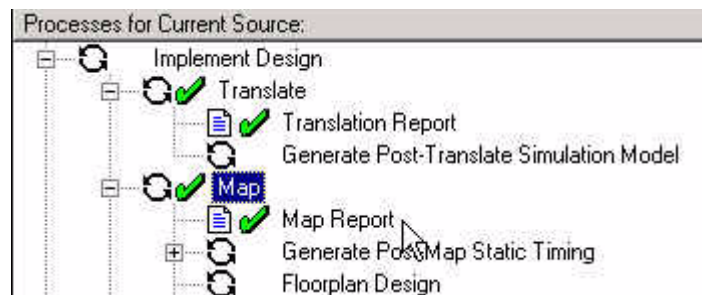


Figure 4-12 Translation Report and Map Report

5. Review the report for Warnings, Errors, and Information (INFO).

Using the Floorplanner

The Floorplanner is a graphical placement tool that gives you control over placing a design into a target FPGA using a "drag and drop" paradigm with the mouse pointer. The Floorplanner is specifically intended to assist those users who require some degree of handcrafting for their designs. You must understand both the details of the device architectures and how floorplanning can be used to refine a design.

The Floorplanner displays a hierarchical representation of the design in the Design Hierarchy window using hierarchy structure lines and colors to distinguish the different hierarchical levels. The Floorplan window displays the floorplan of the target device into which you place logic from the hierarchy. Logic symbols represent each level of hierarchy in the Design Hierarchy window. You can modify that hierarchy in the Floorplanner without changing the original design. You use the mouse to select the logic from the Design Hierarchy window and place it in the FPGA represented in the Floorplan window.

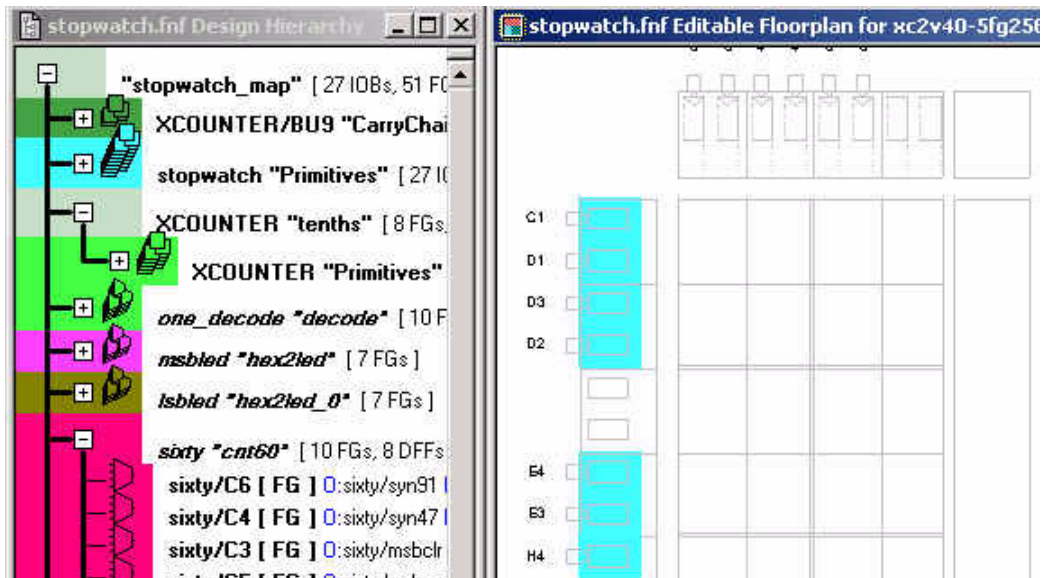


Figure 4-13 Design Hierarchy and Floorplan Windows

With the Floorplanner, you can floorplan your design prior to or after running PAR. In an iterative design flow, you floorplan and place and route, interactively. You can modify the logic placement in the Floorplan window as often as necessary to achieve your design goals. You can save the iterations of your floorplanned design to use later as a constraints file for MAP.

Alternatively, you can invoke the Floorplanner after running the place and route tools to view and possibly improve the results of the automatic implementation. Floorplanning is an optional methodology to help you improve performance and density of a fully, automatically placed and routed design. Floorplanning is particularly useful on structured designs and data path logic. With the Floorplanner, you see where to place logic in the floorplan for optimal results, placing data paths exactly at the desired location on the die.

In this section, the Floorplanner is used to make IOB assignments. The Floorplanner will edit the UCF file by adding the newly created placement constraints. The placement constraints you create in the Floorplanner take precedence over existing constraints in the UCF. You are going to focus on locking down several IOs into the UCF.

1. Expand the MAP tree.
2. Launch Floorplanner on the post-map design by double-clicking on the Floorplan Design.

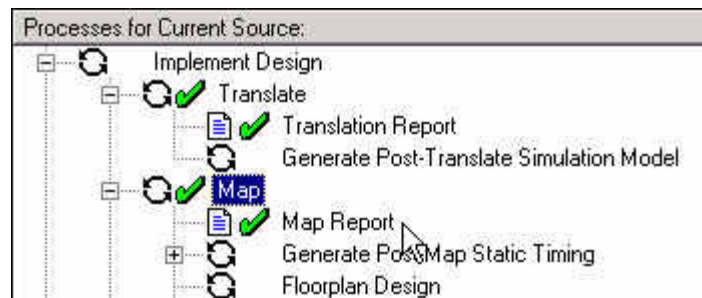


Figure 4-14 Launching Floorplanner

3. You are going to place some IOs in the Floorplan Window. In the Hierarchical Design Window, expand the stopwatch “Primitives” tree.

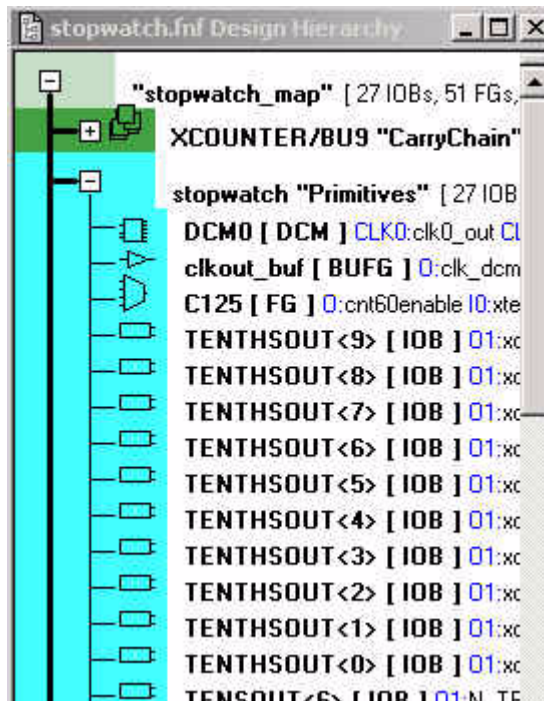


Figure 4-15 Stopwatch Primitives Hierarchy

4. Drag and drop the following nets to the specific locations:

Tenthout<9> -> A7
 Tenthout<8> -> B7
 Tenthout<7> -> A8
 Tenthout<6> -> B8
 Tenthout<5> -> C8
 Tenthout<4> -> D8
 Tenthout<3> -> D9
 Tenthout<2> -> C9
 Tenthout<1> -> B9
 Tenthout<0> -> A9
 Strtstop -> B12
 Reset -> A12
 CLK -> A10

Note: If you have already completed the Schematic chapter of this tutorial, the Strtstop, Reset, and CLK pins are already locked down.

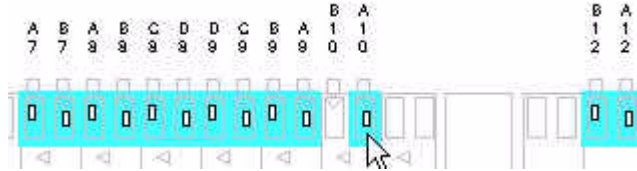


Figure 4-16 Net Locations

5. Once the pins are locked down, select **File** → **Use UCF Flow**. This will enable the UCF flow and the pin locks to be written to stopwatch.ucf.
6. Save the UCF by writing out the constraints, **File** → **Write Constraints**, and select the stopwatch.ucf file and press Save.

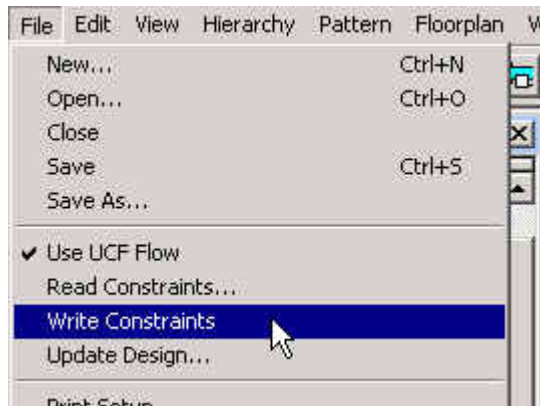


Figure 4-17 Write Constraints Menu Option

7. Exit Floorplanner by **File** → **Exit** and select Yes and Save to the following dialogs for stopwatch.ucf.
8. If you get a dialog box, asking to Reset the Implement Design process..., select Reset to reset the process.
9. Now, you will continue with the implementation, but translate and map will need to be re-run. Right-click MAP and select Re-run All.

Using Timing Analysis to Evaluate Block Delays After Mapping

After the design is mapped, you can use the Logic Level Timing Report to evaluate the logical paths in the design. Because the design is not placed and routed yet, actual routing delay information is not yet available. The timing report describes the logical block delays and estimated routing delays. The net delays that are provided are based on an optimal distance between blocks (also referred to as *unplaced floors*).

Estimating Timing Goals with 50/50 Rule

You can get a preliminary idea of how realistic your timing goals are by evaluating a design after the map stage. A rough guideline (known as the 50/50 rule) specifies that the block delays in any single path make up approximately 50% of the total path delay after the design is routed. For example, a path with 10ns of block delay should meet a 20ns timing constraint after it is placed and routed. If your design is extremely dense, the Logic Level Timing Report provides a summary analysis of your timing constraints based on block delays and estimates of route delays that can help to determine if your timing constraints are going to be met. This report is produced after Map and prior to PAR (Place And Route).

Report Paths in Timing Constraints Option

Because timing constraints were defined for this tutorial design, the Report Paths in Timing Constraints option was selected. This option forces the Logic Level Timing Report to provide a period and path analysis on the constraints specified. Taking a look at the report, the period timing constraint is listed on top, as is the minimum period obtained by the tools after mapping. Because the report was limited to one path per timing constraint, you see a breakdown of a single path that contains 4 levels of logic. Notice the percentage of block (logic) delay versus routing delay for this calculation. The unplaced floors listed are estimates (indicated by the letter “e” next to the net delay) based on optimal placement of blocks.

If you do not generate a Logical Level Timing Report, PAR still processes a design based on the relationship between the block delays, floors, and timing specifications for the design. For example, if a PERIOD constraint of 8 ns is specified for a path, and there are block delays of 7 ns and unplaced floor net delays of 3 ns, PAR stops and generates an error message. In this example, PAR fails because it determines that the total delay (10 ns) is greater than the constraint placed on the design (8 ns). Use the Logic Level Timing Report to determine timing violations that may occur prior to running PAR.

So, you are going to open the Logic Level Timing Report and review the PERIOD Constraint that were entered earlier.

1. Make sure the Map tree is expanded and then double-click Generate Post-Map Static Timing Report.
2. To open the Post-Map Static Timing Report, double-click Post-Map Static Timing Report. Timing Analyzer is automatically launched and shows the report.

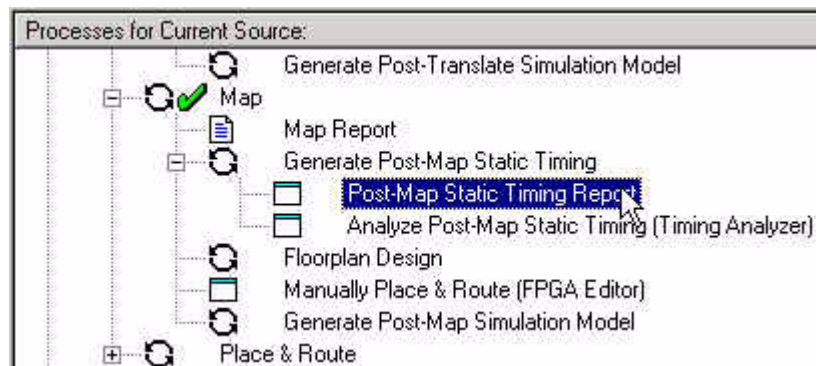


Figure 4-18 Post-Map Static Timing Report

In Timing Analyzer, do we have any timing errors reported?

What is the Minimum Period? _____

Answers: No timing errors and the Min Period should be around 3.087ns.

3. Exit Timing Analyzer by selecting **File** → **Exit**.

Placing and Routing the Design

After the mapped design is evaluated to verify that block delays are reasonable given the design specifications, the design can be placed and routed. The Flow Engine can perform the following place and route algorithms.

- Timing Driven —run PAR with timing constraints specified from within the input netlist or from a constraints file
- Non-Timing Driven —run PAR and ignore all timing constraints

In this tutorial, timing driven placement and timing driven routing are automatically performed by PAR because timing constraints are specified for this design.

1. In the Design Implement tree, run Place and Route by double-clicking Place & Route.
2. Review the reports generated to make sure the place and route process finished as expected. Expanding the Place & Route tree and double-clicking the Place & Route Report does this.

Besides the Place & Route Report, there are two other reports generated by PAR.

Table 4-3 Reports Generated by PAR

Place & Route Report	Provides a device utilization and delay summary. Use this report to verify that the design successfully routed and that all timing constraints were met.
Pad Report	Contains a report of the location of the device pins. Use this report to verify that pins locked down were placed in the correct location.
Asynchronous Delay Report	Lists all nets in the design and the delays of all loads on the net.

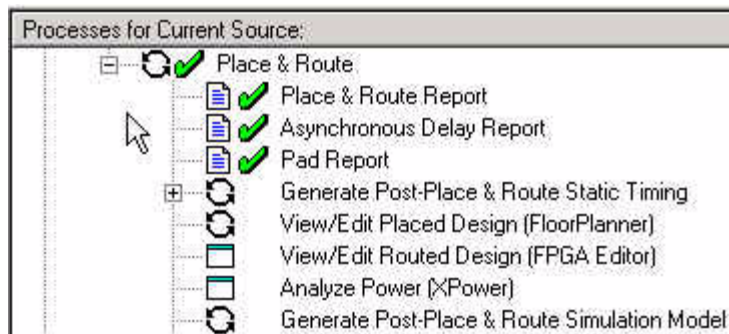


Figure 4-19 Available PAR Reports

Using FPGA Editor to Verify the Place and Route

The FPGA Editor is a graphical application for displaying and configuring Field Programmable Gate Arrays (FPGAs). The FPGA Editor reads from and writes to Native Circuit Description (NCD) files, macro files (NMC), and Physical Constraints Files (PCF).

The following is a list of a few of the functions you can perform on your designs in the FPGA Editor.

- Place and route critical components before running the automatic place and route tools.
- Finish placement and routing if the routing program does not completely route your design.
- Add probes to your design to examine the signal states of the targeted device. Probes are used to route the value of internal nets to an IOB (Input/Output Block) for analysis during the debugging of a device.
- Run the BitGen program and download the resulting bitstream file to the targeted device.
- View and change the nets connected to the capture units of an Integrated Logic Analyzer (ILA) core in your design.

First, launch FPGA Editor and view the actual design layout on the FPGA.

1. In the expanded Place & Route tree, launch FPGA Editor by double-clicking View/Edit Routed Design (FPGA Editor).

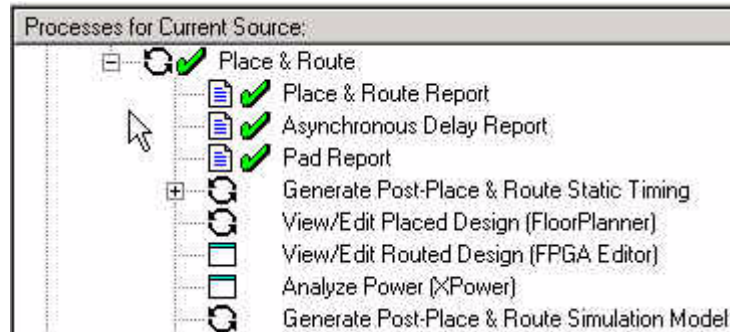


Figure 4-20 View/Edit Routed Design (FPGA Editor) Process

2. After FPGA Editor is open, change the List Window from All Components to All Nets. This allows you to view all of the possible nets in the design.

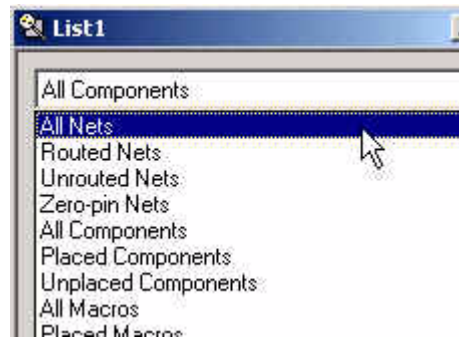


Figure 4-21 List Window in FPGA Editor

3. Select the clk_dcm (Clock) net and see the fanout of the clock net.

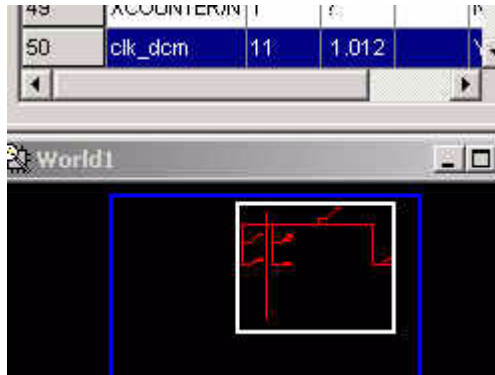


Figure 4-22 Clock Net

4. Exit FPGA Editor by selecting **File** → **Exit**.

Evaluating Post-Layout Timing

After the design is placed and routed, a Post Layout Timing Report is generated by default to verify that the design meets your specified timing goals. This report evaluates the logical block delays and the routing delays. The net delays are now reported as actual routing delays after the place and route process (indicated by the letter “R” next to the net delay).

1. Double-click Generate Post-Place & Route Static Timing to create the report.
2. Expand the Generate Post-Place & Route Timing tree and double-click Post-Place & Route Static Timing Report to open it in Timing Analyzer.

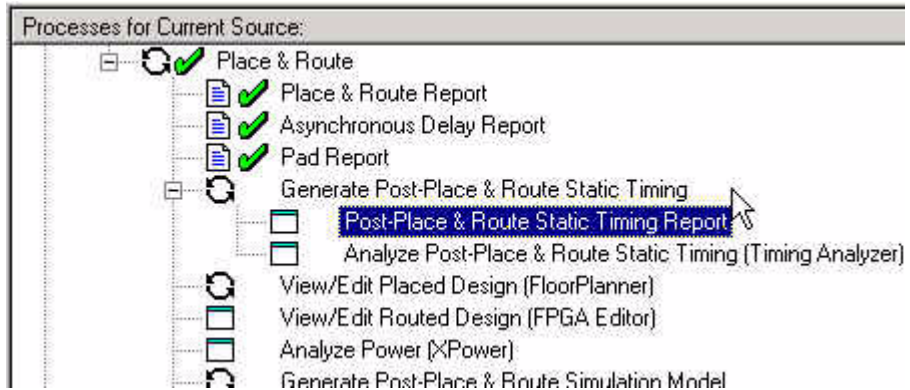


Figure 4-23 Post-Place & Route Static Timing Report

Following is a summary of this report.

- The minimum period value increased due to the actual routing delays.
- After the Map step, logic delay contributed to about 80% of the minimum period attained. The post-layout report indicates that the logical delay value decreased somewhat. The total unplaced floors estimate changed as well. Routing delay after PAR now equals about 31% of the period; a true report of net delays after the place and route step.
- The post-layout result does not necessarily follow the 50/50 rule previously described because the worst case path includes primarily component delays. After the design was mapped, block delays constituted about 80% of the period.

After place and route, the majority of the worst case path is still made up of logic delay. Since total routing delay makes up only a small percentage of the total path delay, spread out across three nets, expecting this to be reduced any further is unrealistic. In general, you can reduce excessive block delays and improve design performance by decreasing the number of logic levels in the design.

Creating Configuration Data

This section explains how to create configuration data. This section includes creating a bitstream for the target device by running the configure step, as follows:

1. Right-click Generate Programming File and select Properties.

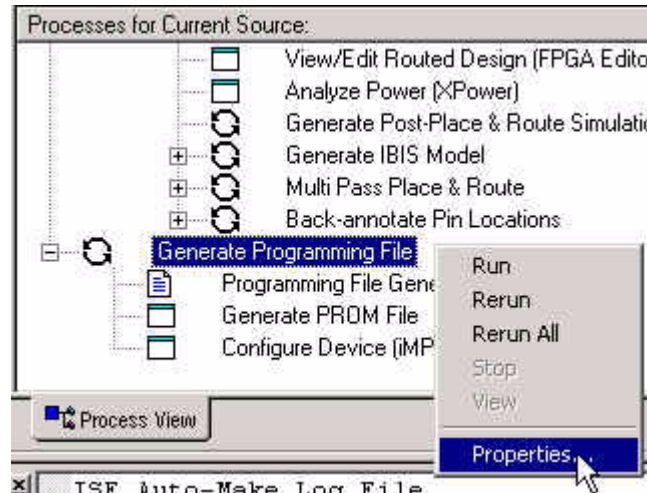


Figure 4-24 Selecting Properties

The Process Properties dialog box appears with several tabs available.

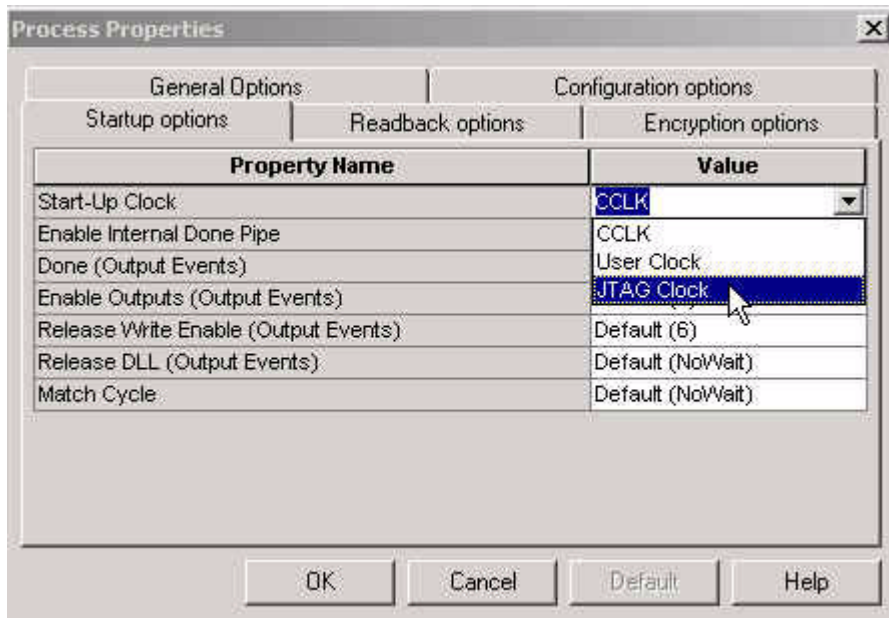


Figure 4-25 Process Properties' Startup Options Tab

2. Select the Startup Options tab, and change the Startup Clock from CCLK to JTAG, since you are going to configure this device via Boundary Scan. This can remain at CCLK, if you were doing Select Map or Serial Slave configuration.
3. Leave the remaining options in the default setting and select **OK** to apply the new properties.
4. Double-click on Generate Programming File to create a bitstream of this design.

The bitstream comes from the BitGen program and creates the *design_name.bit* and *design_name.ll* files (in this tutorial, the *watch.bit* and *watch.ll* files). The *design_name.bit* file is the actual configuration data. The *design_name.ll* file is the logical allocation file that is used during iMPACT to determine the location of the probable points in the design. These files are automatically copied to your working directory.

5. Verify that these files are in this directory. The *design_name.11* file is used to perform device readback with the iMPACT tool. For more information on device readback, please refer to the latest version of the *Watch Design-Hardware Verification Tutorial* (<http://support.xilinx.com/support/techsup/tutorials/index.htm>)
6. Review the Programming File Generation Report by double-clicking the report. Verify that the specified options were used when creating the configuration data.



Figure 4-26 Programming File Generation Report

Using the PROM File Formatter

If you are going to program a single device using iMPACT, all you need is a *design.bit* file. If you are going to program several devices in a daisy chain configuration, or program your devices using a PROM, you must use the PROM File Formatter (PFF) to create a PROM file. The PROM File Formatter accepts any number of bitstreams and creates one or more PROM files containing one or more daisy chain configurations.

1. To start the PROM File Formatter (PFF), double-click on Generate PROM File.

The PFF starts with a default PROM that matches the currently selected (configured) revision. At this point, you can add additional bitstreams to the daisy chain; create additional daisy chains; remove the current bitstream and start over; or immediately save the current PROM file configuration.

The status bar at the bottom of the PFF window displays the PROM format, data format, current PROM size, and percentage of the selected PROM used by the current PROM configuration. The currently selected PROM is an XC17512L. The PFF determined that an XC17512L is the correct PROM because it can hold up to 1,048,576 configuration bits (or 53% full).

The right half of the PFF window is a directory structure used for locating bitstreams. Only files with a .BIT extension are shown in the list. For detailed information on using the PROM File Formatter to create daisy chains or complex PROM configurations, see the *PROM File Formatter Reference/User Guide*. This tutorial describes how to save the default PROM file.

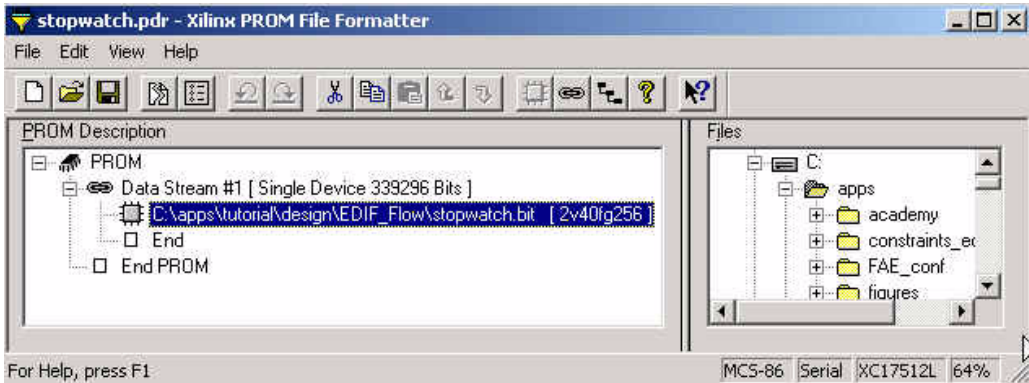


Figure 4-27 PROM File Formatter

2. Select **File** → **PROM Properties** to open the PROM Properties dialog box, shown in the following figure.
3. Select the following options in this dialog box.
 - PROM File Format from the drop-down list
 - PROM Type
 - Number of PROMS used to hold the data

If you have more data than space available in the PROM, you must split the data into several individual PROMs with the Split PROM option. In this case, only a single PROM is needed.

4. Click **OK** to accept the PROM Properties.

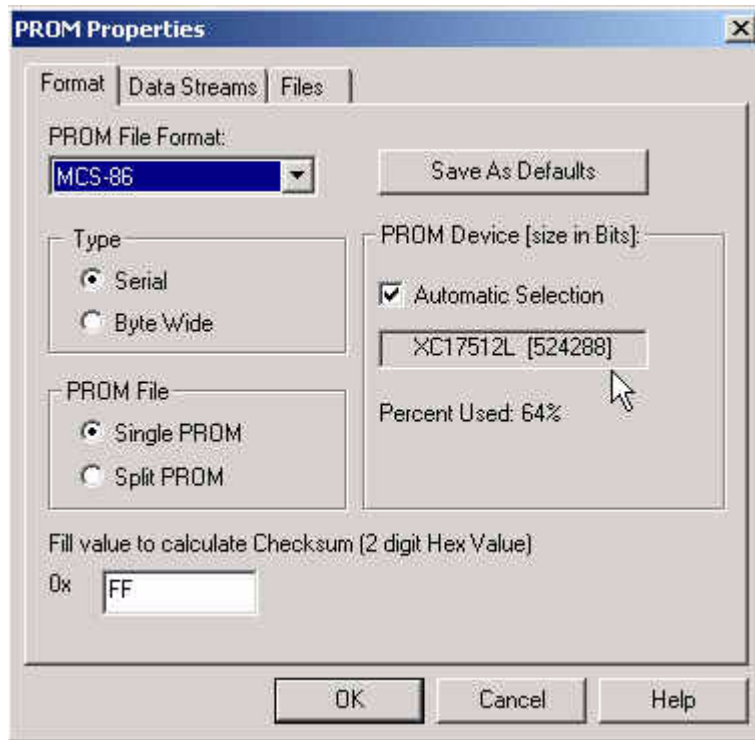


Figure 4-28 PROM Properties Dialog Box

5. Select **File** → **Save Description** to save the PROM file.
6. Specify your working directory as the area where the PROM Description File will be saved. The PROM File Formatter saves both the PROM file (watch.mcs) and a PROM Description File (watch.pdr). The PDR file can be re-opened if any changes are required. Verify that the files exist in your directory.
7. Select **File** → **Exit** to close the PROM File Formatter.

This completes this chapter of the tutorial. For more information on this design flow and implementation methodologies (especially some of the tools/programs that were not covered as part of this tutorial), please reference the online version of the Software Manuals at <http://support.xilinx.com>.

Timing Simulation

Timing simulation uses the block and routing delay information from a routed design to give a more accurate assessment of the behavior of the circuit under worst-case conditions. For this reason, timing simulation is performed after the design has been placed and routed.

This chapter includes the following sections.

- [“Overview of Timing Simulation Flow”](#)
- [“Starting Modelsim”](#)
- [“Adding Signals”](#)
- [“Saving the Simulation”](#)

Overview of Timing Simulation Flow

Timing (post-place and route) simulation is a recommended part of the HDL design flow for Xilinx devices. Timing simulation, also known as back-annotated simulation, uses the detailed timing and design layout information that is available after place and route to create a VHDL or Verilog simulation netlist. This enables simulation of the design, which closely matches the actual device operation.

Required Files

The timing simulation flow requires the following files:

- Design Files (VHDL or Verilog)—The design file is produced by the Xilinx software.
- Stimulus File (VHDL or Verilog)—This is also known as the testbench. You can use the same testbench for functional simulation as well as timing simulation. Also, you can create the testbench with Xilinx HDL Benchner—See the [“Timing Simulation” chapter](#) for information on this flow.
- Modelsim Script File (Optional)—The script file (.do) automates the simulation to a large extent, and makes it easy to re-run the simulation. Alternatively, the commands can be entered one-by-one into the simulator. Xilinx ISE creates the script file needed to run simulation in Modelsim.
- Xilinx Simulation Libraries—For timing simulation, the SIMPRIMS HDL simulation library must be used. Details about this library are provided in the following section.

Xilinx Simulation Libraries

To perform timing simulation of Xilinx designs in any HDL simulator, the SIMPRIM library must be setup correctly. The timing simulation netlist created by Xilinx is composed entirely of instantiated primitives, which are located in the SIMPRIM library. The recommended mapping name for the VHDL SIMPRIM library is SIMPRIM, and for the Verilog SIMPRIM library is SIMPRIMS_VER.

For detailed instructions on compiling these libraries, see Xilinx Answer Record # 2561, which can be accessed as follows:

1. Go to <http://support.xilinx.com>.
2. Enter “2561” in the search box, and check to see that the search engine is pointing to “Answer Records.” Click OK.
3. Click the link to Answer Record # 2561.

Starting Modelsim

Xilinx ISE is fully integrated with any version of the ModelSim Simulator. ISE provides work directory creation, source file compilation, simulation initialization, and simulation property control.

Specifying Simulation Process Properties

In the Sources in Project window, select stopwatch_tb.vhd (stopwatch_tb.tf for Verilog). This displays the ModelSim Simulator Process:

1. Click the + to expand this process.
2. Select and right-click Simulate Post-Place & Route VHDL (Verilog) Model.
3. Select Properties.

The following properties are shown, and can be edited from the pop-up GUI (as shown in Figure 5-1):

Simulation Properties

- Custom Do File—Use this option to specify a different .do run script.
- Use Automatic Do File— If this option is unchecked, Modelsim will start but not automatically run the processes required to simulate the design. You must manually run the .do file from Modelsim, or enter the commands one-by-one to run simulation.
- Simulation Run Time—Use this option to specify the default time for which simulation is run.
- Simulation Resolution—This property is set to 1 ps by default, but can be changed as required.
- Design Unit Name—Use this property to specify the top level model to be loaded in ModelSim. Use this property if the top level entity, configuration, or module is named something other than the testbench name.

Display Properties

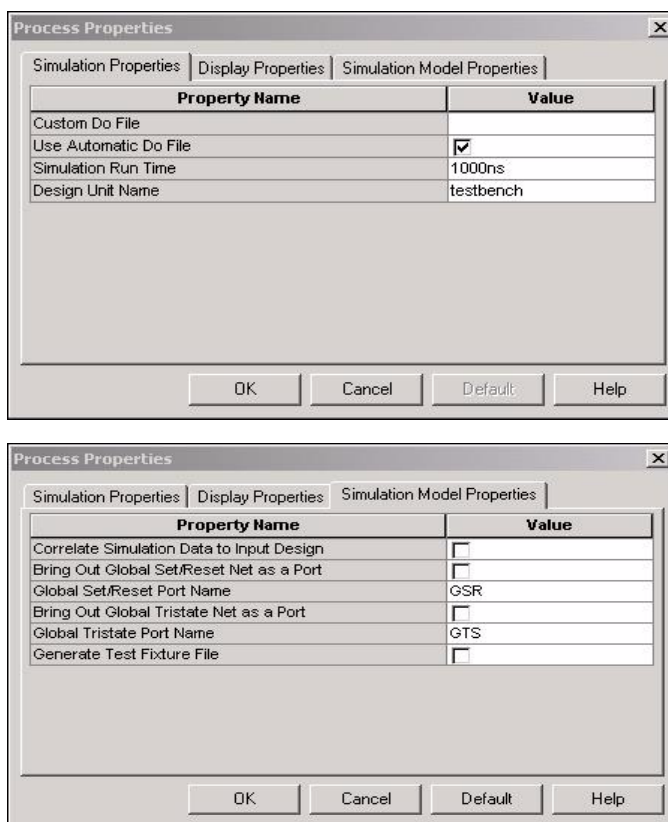
This tab gives you control over the MTI simulation windows. These are the windows open by default, when timing simulation is launched from Foundation ISE. By default, the Signal, Structure, and Wave windows are launched. For more details on Modelsim Simulator windows, refer to the *ModelSim User Manual*.

Simulation Model Properties

- Correlate Simulation Data to Input Design—By selecting this property, you instruct the Xilinx post-place and route netlist generation tools to append the timing details to the input design (post NGDBUILD design). The advantage of using this option is that the user-defined signal names are preserved. The disadvantage is that the user design, rather than the physical (post-place and route) design, is used. Therefore, if there are any errors introduced by the place and route tools, they will not be caught.
- Bring Out Global Set/Reset Net as a Port—Use this option to create an external port in the simulation netlist that will allow you to control of the power-on-reset from a port.
- Global Set/Reset Port Name— Default is GSR.
- Bring Out Global Tristate Net as a Port.
- Global Tristate Port Name— Default is GTS.
- Generate Test Fixture/Testbench File—Use this option to create a testbench.

The following options will show up if the Advanced Process Settings are enabled in Project Navigator.

- Retain Hierarchy in Netlist.
- ROC Pulse Width—Use this option to set the duration of the Global Reset on Configuration Pulse Width. The default is 100 ns.
- TOC Pulse Width—Use this option to set the duration of the Global Tristate on Configuration Pulse Width. The default is 1 ns.

**Figure 5-1 Simulation Process Properties**

Performing Simulation

Once you have set the process properties, Modelsim you can invoked. To start the functional simulation, double-click Simulation Functional VHDL or Verilog Simulation.

ModelSim will now create the work directory, compile the source files, load the design, and run simulation for the time specified.

There are two basic steps to simulate your design:

1. Adding Signals
2. Saving the Simulation

There are several different ways to perform each of these steps. These methods are discussed briefly in the following sections.

Adding Signals

To view signals during the simulation, you must first add them to the Wave window. Project Navigator automatically adds all of the top level ports to the Wave window. Additional signals are displayed in the Signal window based upon the selected structure in the Structure window.

There are two basic methods for adding signals to the Simulator Wave window.

- Drag and drop from the Signal window
- Select **View** → **Wave** → **Selected Signals** from the Signal Window

The following procedure explains how to add additional signals in the design hierarchy. For the purpose of example, add the smallcnt output flip-flops used to count.

1. In the Structure window, click the + next to uut:stopwatch(structure).
 - *uut* is the instance in the testbench.
 - *stopwatch* is the component/entity name.
 - *structure* is the architecture name.

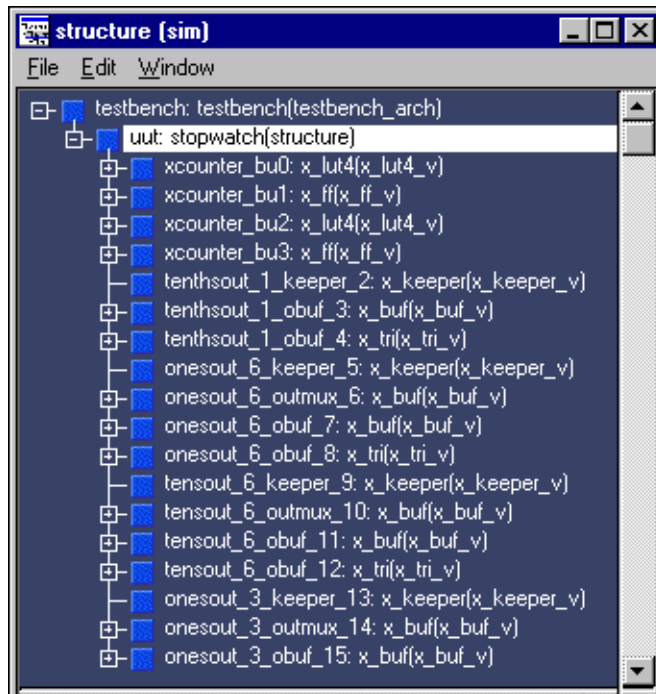


Figure 5-2 Structure Window

2. Select sixty_lsbcount_qoutsig_0 in the Signals window. Notice that the signals listed in the Signal window are updated.
3. Click and drag sixty_lsbcount_qoutsig_0 from the Signal window to the Wave window.
4. Select sixty_lsbcount_qoutsig_1 in the Signal window, and select **View → Wave → Selected Signals** to add the signal to the Wave window. Notice that the waveforms have not been drawn for these signals.

5. There are two ways to add the waveforms for these signals: continue with the simulation, and restart the simulation.
 - To continue to run the simulation, click the Continue Run icon on the toolbar.



Figure 5-3 Continue Run Icon

- To restart the simulation, click the Restart Simulation icon. The Restart dialog box opens. Click Restart.



Figure 5-4 Restart Icon

6. Click the Continue Run icon to re-run the simulation.

Saving the Simulation

The ModelSim Simulator gives you the ability to save the signals list in the Wave window. This can be important when additional signals or stimulus have been added, and the simulation must be restarted. You can easily load the saved signals list each time the simulation is started.

1. In the Wave window, select **File** → **Save Format**.
2. In the Save Format dialog box, change wave.do to sec_signal.do.

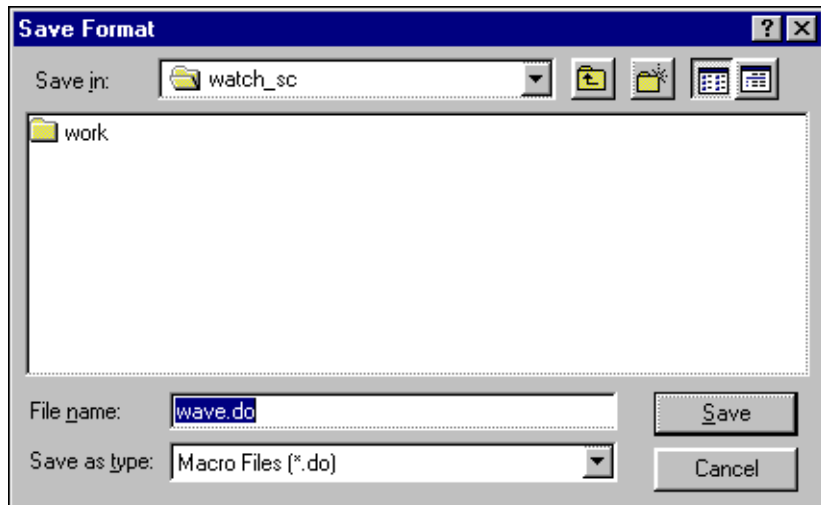


Figure 5-5 Save Format Dialog Box

3. Click **Save** to close the dialog box.
4. Close the Simulator.

In addition, the simulation results that appear in the waveform are also saved. These are saved in a file called `vsim.wlf`, which is produced by default in the Modelsim project directory (which is the same as the ISE project directory for ISE-MTI projects). If this file needs to be preserved, it must be copied or renamed, otherwise it will be overwritten upon restart of simulation.

To view the contents of this file later, type the following commands one-by-one at the ModelSim prompt:

```
Modelsim> vsim -view vsim.wlf  
vsim> view wave  
vsim> add wave *
```

The simulation output will then appear in the Wave window.

iMPACT Tutorial

This chapter takes you on a tour of Xilinx's next generation device programming tool. iMPACT combines the functionality of the existing JTAG Programmer and Hardware Debugger programs as well as the Coolrunner download program into a single multi-use tool.

This is the first and only chapter in the ["IMPACT Flow."](#) This is an option chapter for all other tutorial flows.

This tutorial contains the following sections:

- ["Device Support"](#)
- ["Download Cable Support"](#)
- ["Configuration Mode Support"](#)
- ["Starting the Software"](#)
- ["Connecting to a Cable"](#)
- ["Boundary Scan Configuration Mode"](#)
- ["Creating a SVF or STAPL File"](#)
- ["Slave Serial Configuration Mode"](#)
- ["Select MAP Configuration Mode"](#)

Device Support

The following devices are supported.

- Virtex™/-II/-II PRO/-E
- Spartan™/-II/-IIE/XL
- XC4000™E/L/EX/XL/XLA
- CoolRunner™XPLA3/-II
- XC9500™/XL/XV
- XC18V01
- XC18V02
- XC18V04
- XC18V256
- XC18V512

Download Cable Support

Parallel Cable III

The Parallel Cable connects to the parallel port and can be used to facilitate Slave Serial and Boundary Scan functionality.

Multilink Cable

The Multilink cable connects to the USB port or the RS232 serial port and can be used to facilitate Slave serial, Select MAP, and Boundary Scan functionality.

Configuration Mode Support

Impact currently supports three configuration modes:

- Boundary Scan —FPGAs, CPLDs, and SPROMs
- Slave Serial—FPGAs
- Select Map—FPGAs (Virtex™/-II/-II PRO/-E and Spartan™/-II/-IIE)

Starting the Software

This section describes how to start the iMPACT software from ISE and shows how to run it stand-alone.

Opening iMPACT from the Project Navigator

First set the Properties for iMPACT which allows you to select the configuration mode and the configuration file. By default, iMPACT will open in Boundary Scan Mode and use the configuration file from the current project. Once iMPACT has started, the configuration mode can be changed and new configuration files can be selected, but setting the properties will ensure that iMPACT opens in the correct Mode and uses the correct configuration file. To set the properties, right-click on Configure Device (iMPACT) in the Processes for Current Source Window and select Properties (see [Figure 6-1](#)).

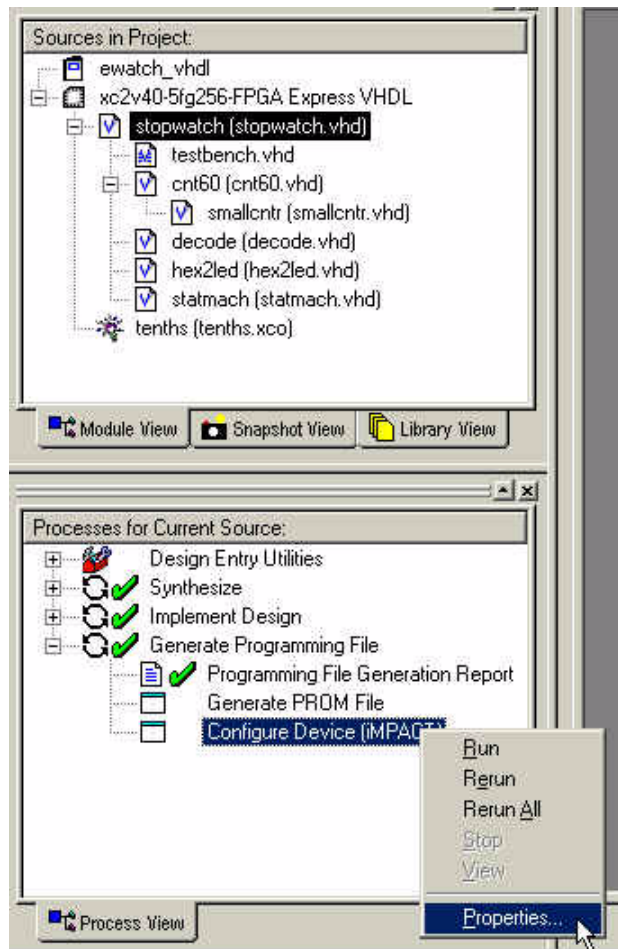


Figure 6-1 Selecting iMPACT Properties in the Processes Window

After selecting Properties, the window displays as shown in the following figure.

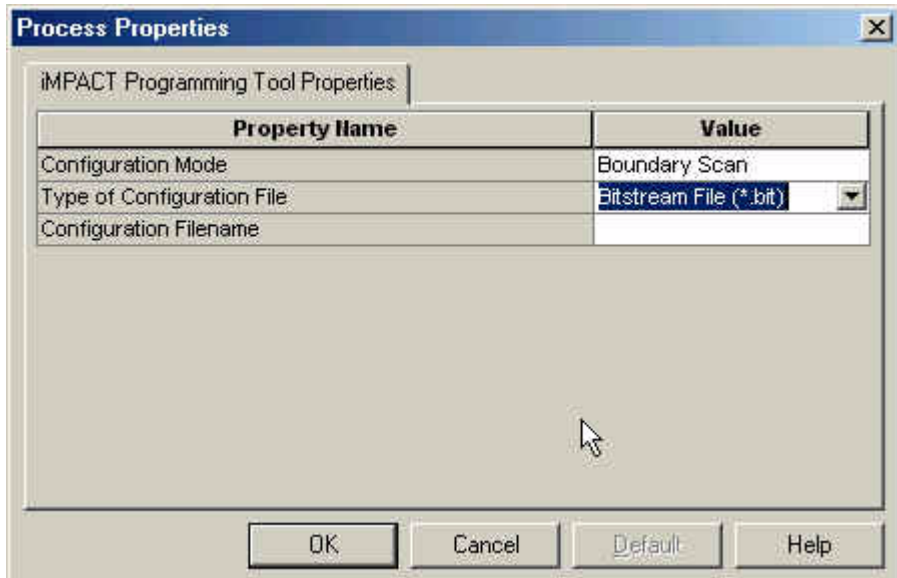


Figure 6-2 iIMPACT Process Properties

From this window, select the Configuration Mode, the Type of Configuration File, and the Configuration Filename. If Bitstream File is selected as the type, the Configuration Filename can be left blank and iIMPACT automatically loads the bit file from the current project. In the [“Creating Configuration Data” section of the “Design Implementation” chapter](#), the Start-Up Clock was set to JTAG Clock because you are going to configure the device through Boundary Scan and so the Properties shown in [Figure 6-2](#) should be selected.

To start iIMPACT from ISE, double-click on Configure Device (iIMPACT) in the Processes for Current Source window (see the following figure).

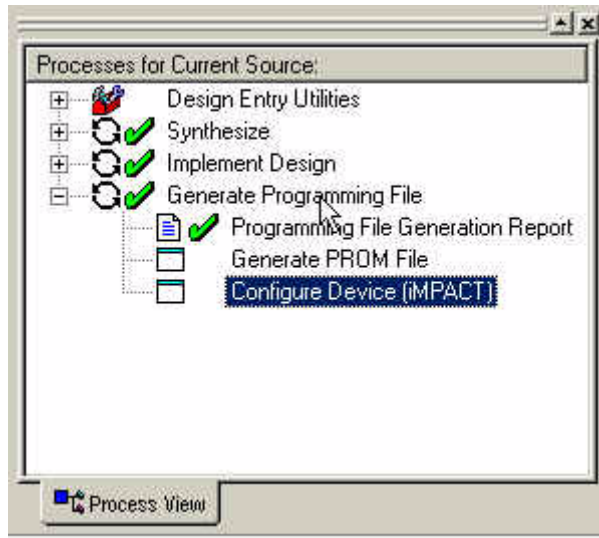


Figure 6-3 Opening iMPACT from ISE

Note: When opening iMPACT from ISE, iMPACT automatically connects to the cable. Make sure that the cable is connected to the computer and that the VCC and GND pins on the cable are connected to VCC and GND on the board. Power must be supplied to the cable before connection can be established.

If the iMPACT properties are set as shown in [Figure 6-2](#), the initial window will look similar to what is shown in [Figure 6-4](#). The device has been added and the configuration file from the project has been applied to the device.

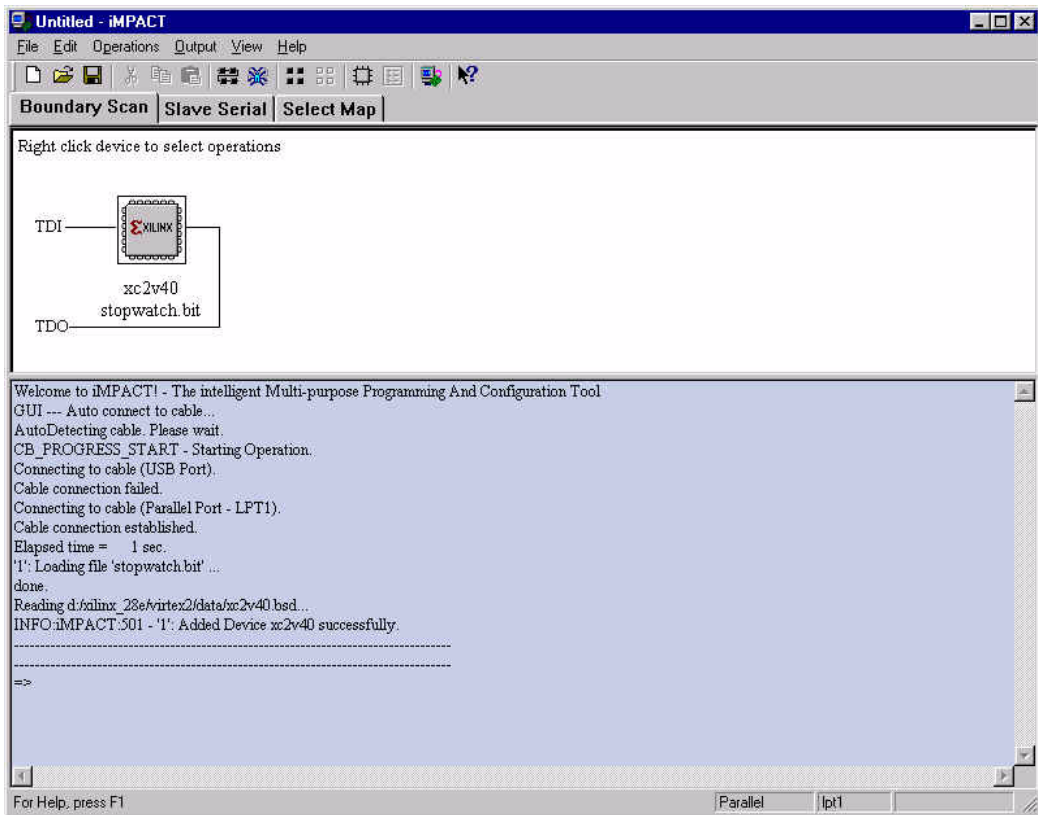


Figure 6-4 Initial Window after Opening iMPACT from ISE

At this point, a single XC2V40 can be configured with stopwatch.bit. If the JTAG pins are connected to the XC2V40 and the XC2V40 is the only device in the chain, right-click the device and select Program. Because it is unlikely that you will have just a XC2V40 that you would like to program, the remainder of the tutorial gives a general overview of how to use iMPACT. The examples use a variety of different devices, not just the XC2V40. Please continue reading through this tutorial to learn how to program multiple devices and how to use the other Configuration Modes.

Opening iMPACT stand-alone

To open iMPACT without going through an ISE project, use one of the following methods.

- PC — Click **Start** → **Xilinx ISE 4** → **Accessories** → **iMPACT**.
- PC or UNIX—Type impact at a command prompt.

When iMPACT is run stand-alone, the initial window appears as shown in [Figure 6-5](#). From this window, you need to select the configuration mode and then add the devices that you would like to program. Refer to the Boundary Scan, Slave Serial, or Select Map section of this tutorial for information on how to use each configuration mode.

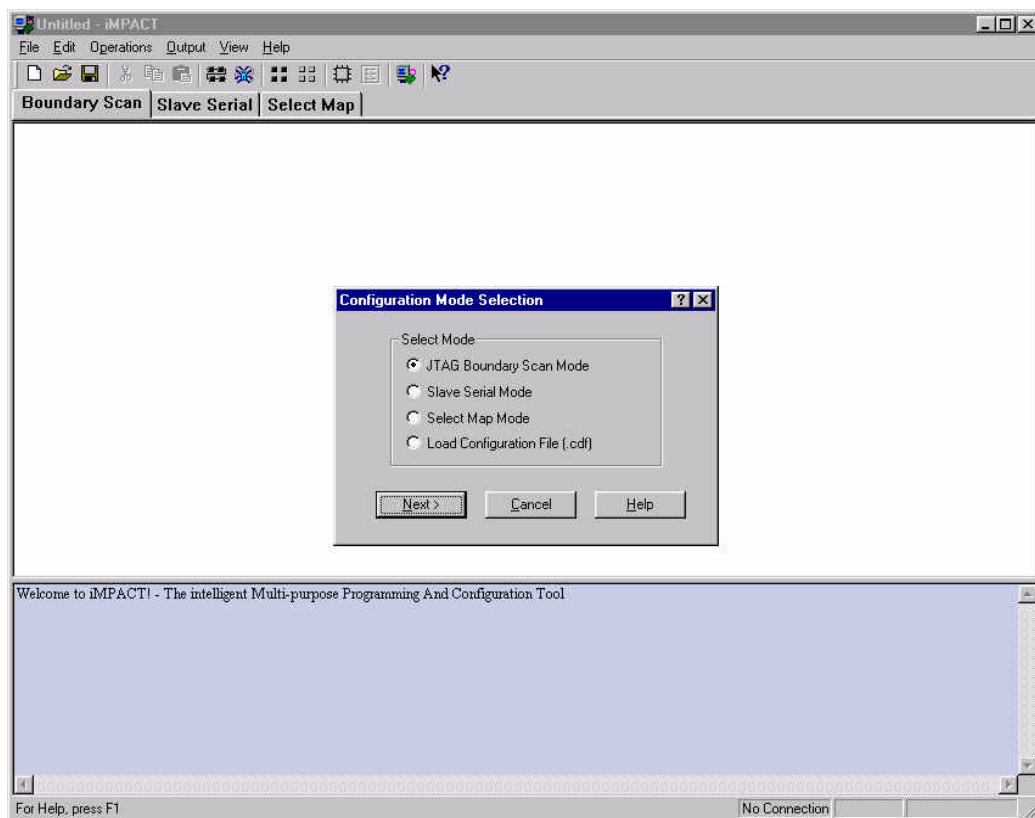


Figure 6-5 Initial Window When Opening iMPACT Stand-Alone

Connecting to a Cable

A cable connection must be established before operations can be performed on a device. When opening iMPACT from ISE or when one of the modes is selected from the window shown in [Figure 6-5](#), the software automatically connects to a cable. If a cable connection has been established, skip this section of the tutorial. If a connection has not been established, right-click in a blank portion of the iMPACT window and select either Cable Auto Connect or Cable Setup (see [Figure 6-6](#)). Cable Auto Connect will force the software to search every port for a connection. Cable Setup allows you to select the cable and the port to which the cable is connected.

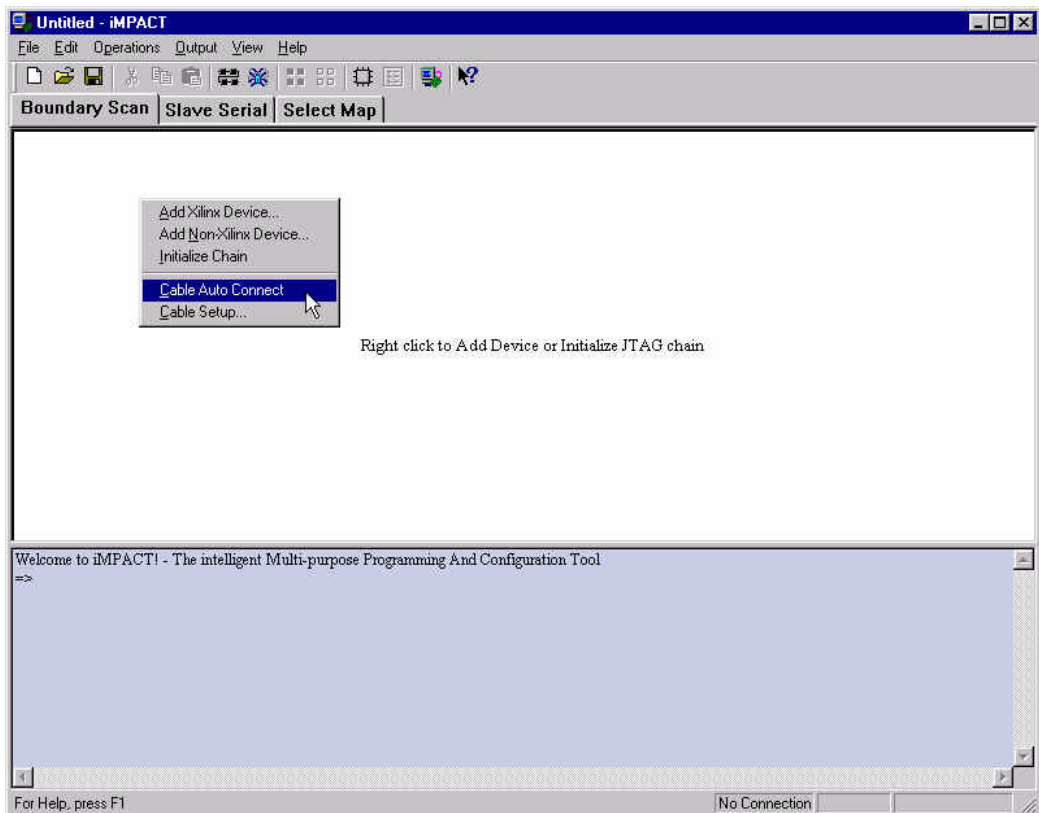


Figure 6-6 Selecting the Cable Connection Method

In iIMPACT the log window shows each port being searched for a connection. [Figure 6-7](#) shows an example where the cable autodetection failed because no cable was connected to the machine. Notice that all eight ports on the machine were searched.

```
Welcome to iIMPACT! - The intelligent Multi-purpose Programming And Configuration Tool
Select Map Mode selected
GUI --- Auto connect to cable...
AutoDetecting cable. Please wait.
Connecting to cable (USB Port).
Cable connection failed.
Connecting to cable (Parallel Port - LPT1).
Cable connection failed.
Connecting to cable (Parallel Port - LPT2).
Cable connection failed.
Connecting to cable (Parallel Port - LPT3).
Cable connection failed.
Connecting to cable (COM1 Port).
Cable connection failed.
Connecting to cable (COM2 Port).
Cable connection failed.
Connecting to cable (COM3 Port).
Cable connection failed.
Connecting to cable (COM4 Port).
Cable connection failed.
=> Cable autodetection failed.
```

Figure 6-7 Failed Attempt to Establish Cable Connection

If a cable is connected to the system and the cable autodetection fails, use the following steps to debug:

1. Verify that the VCC and GND pins of the cable are connected to VCC and GND on the board and make sure that the power supply for the board is turned on.
2. If a connection was previously established with another cable or if the configuration mode has changed, terminate the previous connection by selecting **Output** → **Cable Disconnect** from the menu at the top of the iIMPACT window.
3. Try performing a cable reset by selecting **Output** → **Cable Reset**.
4. Check the connection to the port on the computer and try another port if possible.
5. Shut down the software and reopen it.

6. Verify that the drivers for the cables were installed. Open the fileset.txt file that is located in the directory where the software was installed. The following lines should be in this file:

<Date of install> <Time> <Year>:: summary=MultiLINX Cable Driver

<Date of install> <Time> <Year>:: summary=Parallel Cable III Driver

If these lines are not present, the drivers were not installed. They can be installed by reinstalling the software or by installing the Webpack Programmer.

7. If these suggestions do not help, open at webcase with Xilinx Technical Support at <http://support.xilinx.com>.

Boundary Scan Configuration Mode

Boundary Scan Configuration mode allows you to perform Boundary Scan Operations on any chain comprising JTAG compliant devices. The chain can consist of both Xilinx and non-Xilinx devices, but limited operations will be available for non-Xilinx devices. To perform operations, the cable must be connected and the JTAG pins, TDI, TCK, TMS, and TDO need to be connected from the cable to the board.

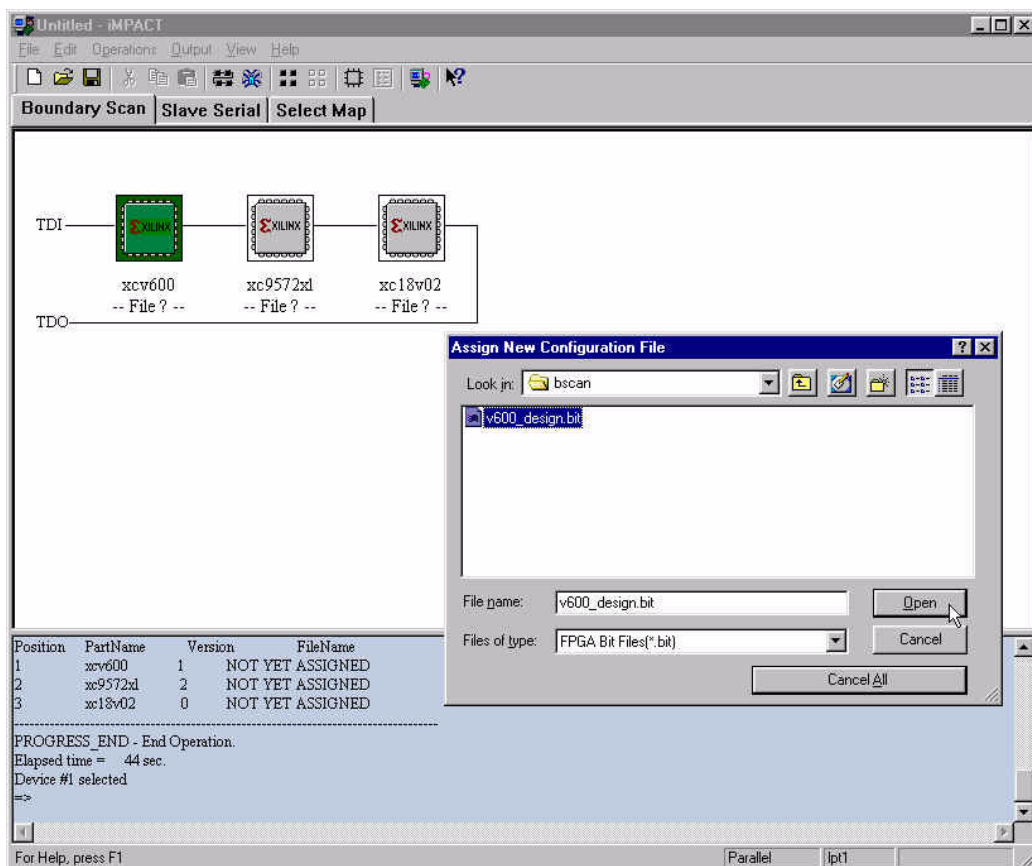
The boundary scan chain that is created in the software must match the chain on the board. If the chain consists of eight devices, but only one of them is going to be configured, all eight devices must be added to the chain in the iMPACT window.

Automatically Creating the Chain

To automatically create the chain, right-click on an empty space in the iMPACT window and select Initialize Chain (see [Figure 6-6](#)).

iMPACT will pass data through the devices and automatically identify the size and composition of the boundary scan chain. Any supported Xilinx device will be recognized and labeled and any other device will be labeled as unknown. The software will then highlight each device in the chain and prompt you for a configuration file.

Figure 6-8 shows an example where the chain consists of three Xilinx devices, a FPGA, a CPLD, and a PROM.

**Figure 6-8 Example Result from Performing Initialize Chain**

Manually Creating the Chain

The chain can be manually created or modified as well. To do this, right-click on an empty space in the iMPACT window and select Add Xilinx Device or Add Non-Xilinx device (see [Figure 6-10](#)). This allows you to add devices one at a time. The device is added where the large cursor is positioned. So, to add a device between existing devices just click on the line between them and then add the new device.

Manually adding devices is useful when creating a chain that is used to generate an SVF or STAPL file, but Initialize Chain should be used whenever possible. Initializing the chain verifies that the chain is set up correctly and that iMPACT can correctly identify all of the devices.

Assigning Configuration Files

After initializing a chain or adding a device, the software prompts you for a configuration file (see [Figure 6-8](#)). This is the file that will be used to program the device. There are several types of configuration files. A Bitstream file (*.bit) is used to configure an FPGA. A JEDEC file (*.jed) is used to configure a CPLD. A PROM file (*.mcs, .exo, .hex, or .tek) is used to configure a PROM.

If a configuration file is not available, a Boundary Scan Description File (BSDL or BSD) file can be applied instead. The BSDL file provides the software with necessary Boundary Scan information that allows a subset of the Boundary Scan Operations to be available for that device. To select a BSDL file, change the file type to *.BSD in the Assign New Configuration File window and browse to the BSDL file (see [Figure 6-9](#)). BSDL files for Xilinx devices are located in the \$XILINX\device\data directories. For example, if the software is installed in c:\xilinx, the BSDL file for a Virtex device is in c:\xilinx\virtex\data.

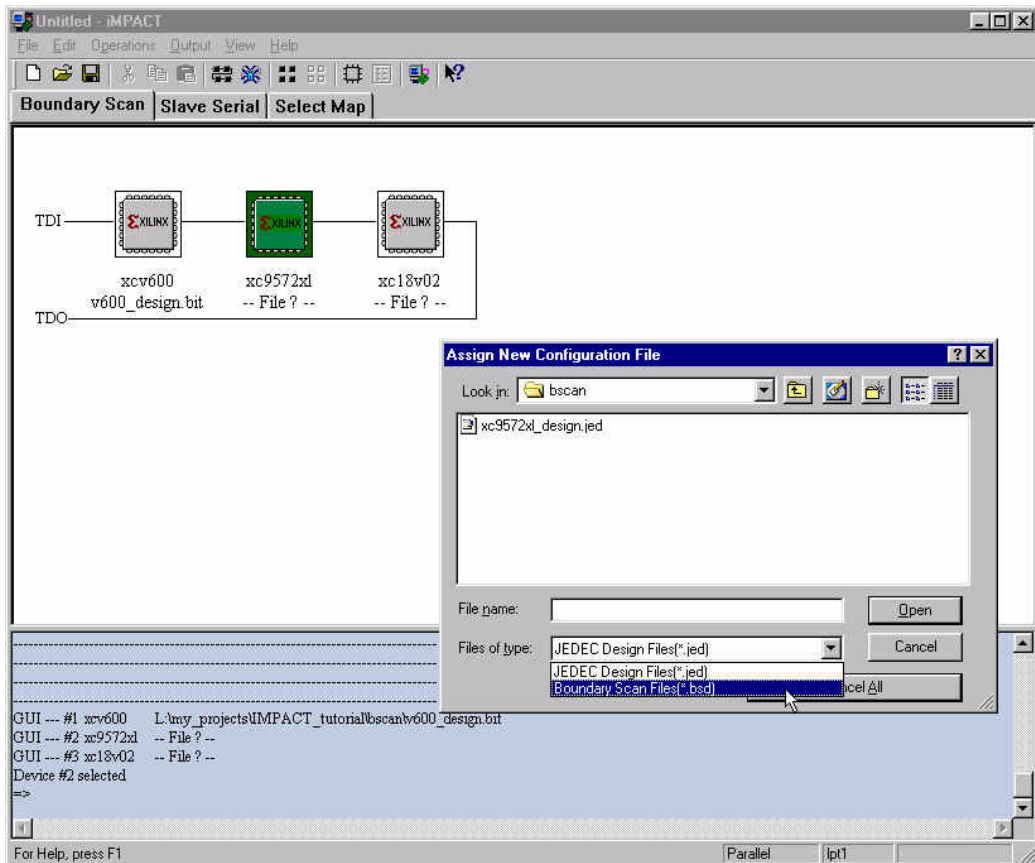


Figure 6-9 Selecting a BSDL File

For non-Xilinx devices, a BSDL or BIT file must be applied. The BSDL file can typically be obtained from the vendor of the device. If a BSDL file cannot be obtained, iIMPACT can create a generic BSDL file. When a non-Xilinx device is added, iIMPACT asks for a BSDL or BIT file for the device (see [Figure 6-10](#)). If yes, then you can browse to the file. If no, iIMPACT will ask you for the device name and the Instruction Register Length (see [Figure 6-11](#)). This minimal amount of information allows iIMPACT to create a generic BSDL file that will allow the device to be put in BYPASS or HIGHZ. Check with the Vendor of the device to obtain the Instruction Register Length.



Figure 6-10 Unknown Device Query



Figure 6-11 Defining an Unknown Device

Saving the Chain Description

Once the chain has been fully described, it can be saved for later use. This prevents you from having to redefine the chain each time the iMPACT software is started. To do this, select **File** → **Save** or **Save As**. This selection creates a Chain Description File (*.CDF). To restore the chain when reopening iMPACT, select **File** → **Open** and browse to the CDF file. The CDF file can also be selected in the Process Properties window in ISE (see [Figure 6-2](#)). This operation restores the chain when opening iMPACT from ISE.

Edit Preferences

To edit the preferences for the Boundary Scan Configuration, select **Edit → Preferences**. This selection opens the window shown in [Figure 6-12](#). Click on help for a description of the Preferences.

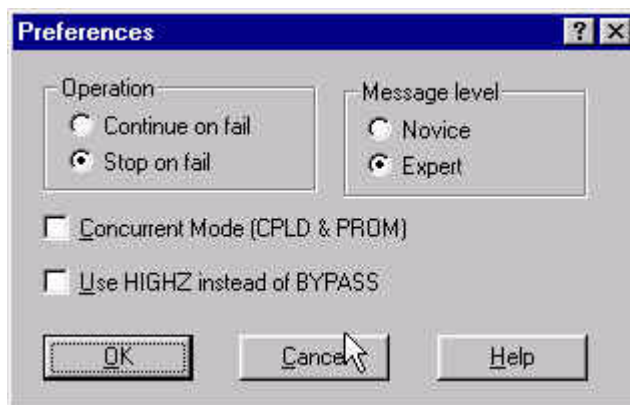


Figure 6-12 Edit Preferences

Available Boundary Scan Operations

The available Boundary Scan Operations vary based on the device and the configuration file that was applied to the device. To see a list of the available options, right-click on any device in the chain. This brings up a window with all of the available options. [Figure 6-13](#) shows the available options for a 9500XL device that has a JEDEC file applied to it.

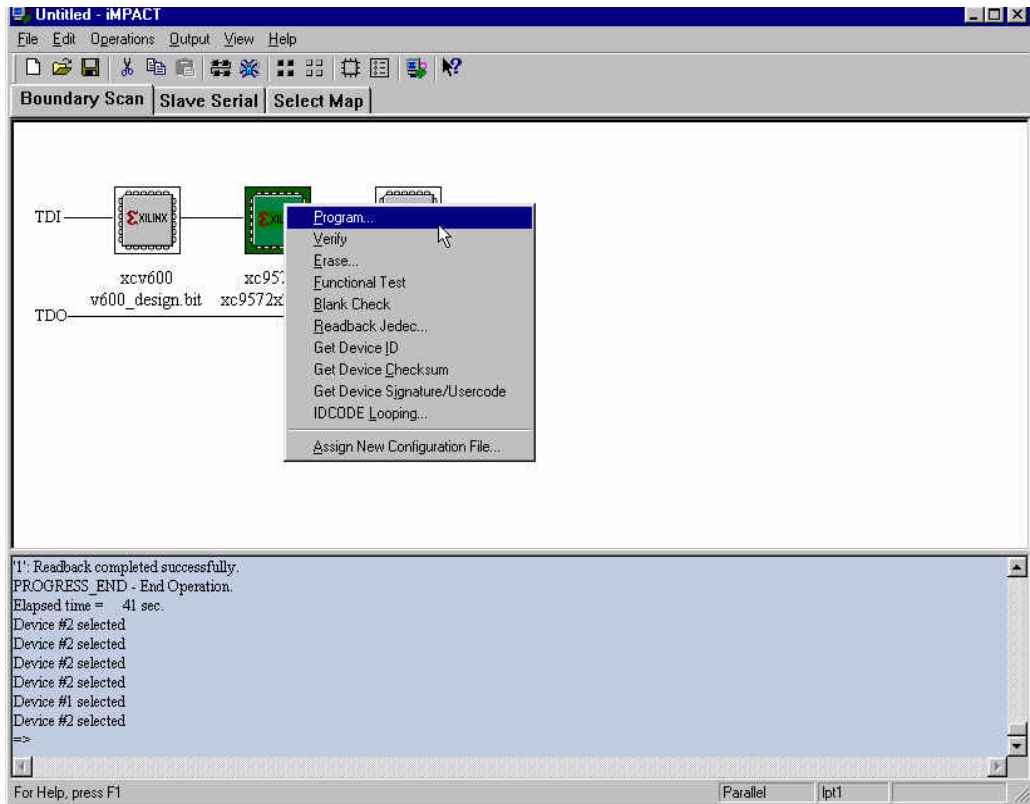


Figure 6-13 Available Boundary Scan Operations for an XC9572XL Device

When the Virtex device is selected, a different set of options is available (see [Figure 6-14](#)).

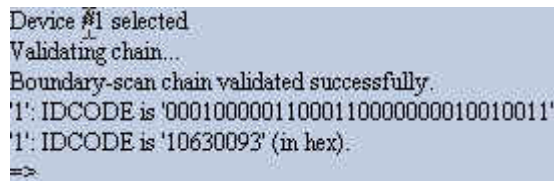


Figure 6-14 Available Boundary Scan Operations for a Virtex Device

Performing Boundary Scan Operations

Boundary Scan operations are performed on one device at a time. When you select a device and perform an operation on that device, all other devices in the chain are automatically placed in BYPASS (or HIGHZ - see [Figure 6-12](#)).

To perform an operation, right-click on a device and then left-click on one of the selections. For instance, when the Virtex Device is right-clicked, the window in [Figure 6-14](#) appears. Then left-click on Get Device ID and the software accesses the IDCODE for this Virtex Device. The result is displayed in the Log Window (see [Figure 6-15](#)).



```
Device #1 selected  
Validating chain...  
Boundary-scan chain validated successfully.  
'1': IDCODE is '00010000011000110000000010010011'  
'1': IDCODE is '10630093' (in hex)  
=>
```

Figure 6-15 Log Window Showing Result of Get Device ID

For another example, if you right-click on the XC9572XL and then left-click on Program (see [Figure 6-13](#)), the Program Option window appears (see [Figure 6-16](#)). You can then select the desired options and click **OK** to begin programming. The Program options vary based on the device.

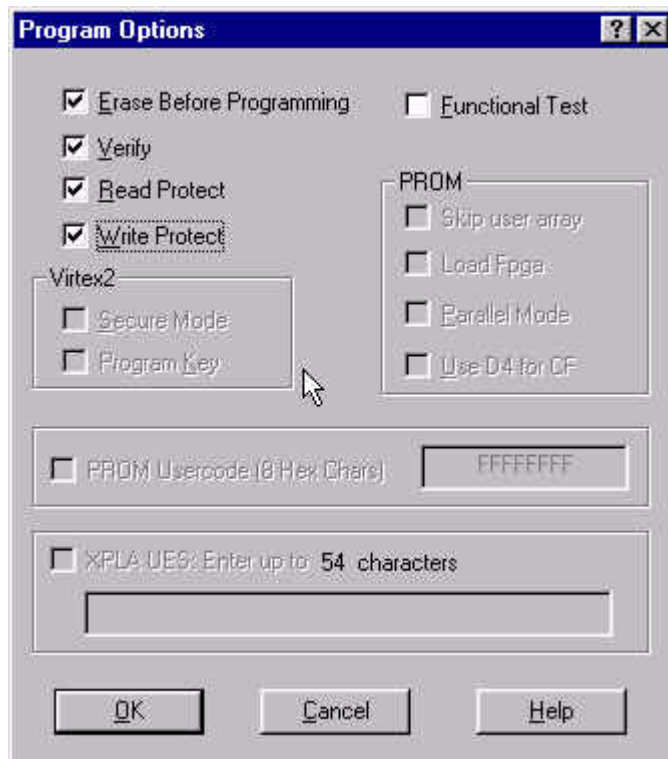


Figure 6-16 Program Options for 9500XL Device

After clicking **OK**, the Program operation begins and an operation status window displays (see [Figure 6-17](#)). At the same time, the log window reports all of the operations being performed.

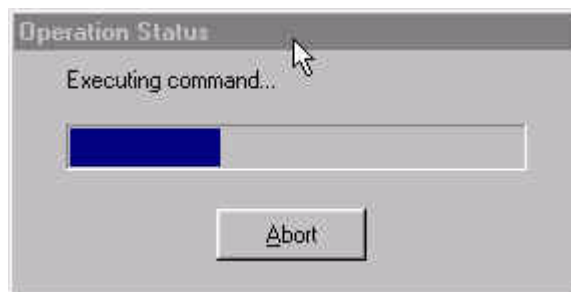


Figure 6-17 Operation Status

When the Program operation completes, a large blue message appears showing that Programming Succeeded (see [Figure 6-18](#)). This message disappears after a couple of seconds.

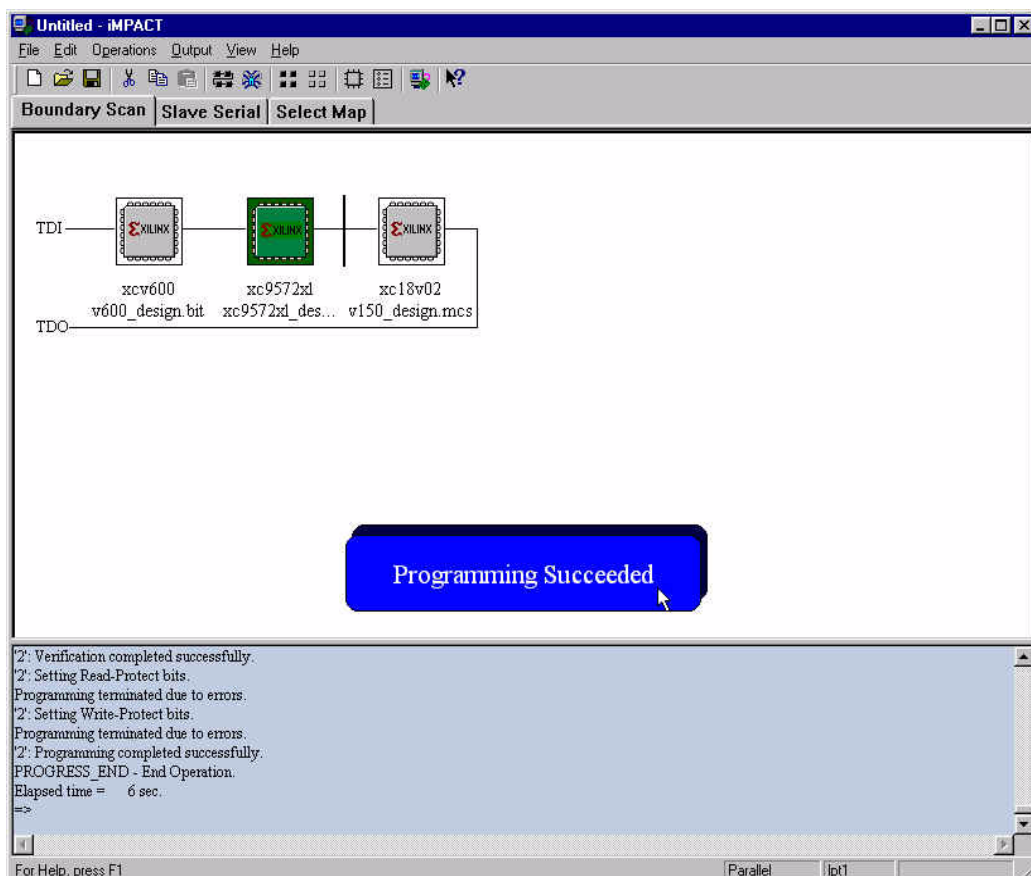


Figure 6-18 Programming Succeeded

The log window also shows that the programming completed successfully and shows all of the operations that were performed (see [Figure 6-19](#)).

```
Device #2 selected
PROGRESS_START - Starting Operation.
Validating chain...
Boundary-scan chain validated successfully.
'2': Putting device in ISP mode...
done.
'2': Erasing device...
done.
'2': Erasure completed successfully.
'2': Programming device...
done.
'2': Putting device in ISP mode...
done.
'2': Verifying device...
done.
'2': Verification completed successfully.
'2': Setting Read-Protect bits.
'2': Setting Write-Protect bits.
Programming terminated due to errors.
'2': Programming completed successfully.
PROGRESS_END - End Operation.
Elapsed time = 6 sec.
=>[
```

Figure 6-19 Log Window Showing Successful Configuration of the CPLD

Operations can continue to be performed in this manner. Select a device and then select the operation. Wait for the previous operation to complete and then perform the next operation.

Troubleshooting Boundary Scan Configuration

When an error occurs during a Boundary Scan operation, first verify that the chain is set up correctly and verify that the software can communicate with the devices. The easiest way to do this is to Initialize the Chain (see the section on [“Automatically Creating the Chain”](#) section). If the chain cannot be initialized, it is likely that the hardware is not set up correctly or the cable is not properly connected. If the chain can be initialized, try performing simple operations. For instance, try getting the Device ID of every device in the chain. If this can be done, then the hardware is set up correctly and the cable is properly connected.

From this point, refer to the JTAG Problem Solver for more debugging information. The JTAG Problem Solver is located at <http://support.xilinx.com/support/troubleshoot/psolvers.htm>.

Creating a SVF or STAPL File

iMPACT currently supports the creation of device programming files in two formats, SVF and STAPL. These programming files contain both programming instructions and the configuration data and are used by ATE machines and embedded controllers to perform Boundary Scan operations. To create an SVF or STAPL file, Boundary Scan mode must be selected. A cable does not need to be connected because no operations are being performed on devices. All of the configuration information is written to the SVF or STAPL file.

Creating the Chain

Before creating the SVF or STAPL file, the Boundary Scan chain must be created. Refer to the section on “[Manually Creating the Chain](#)” [section](#) for Boundary Scan Configuration Mode. The method of adding devices is the same as described in this section.

Select Programming File

After the chain has been fully described, select **Output → Use File → SVF File** or **STAPL File → Create SVF File** or **Create STAPL File** (see [Figure 6-20](#)). These selections access a window that allows you to select a name for your programming file and specify the location for this file. After selecting the name and location, the SVF or STAPL file is ready to be written to.

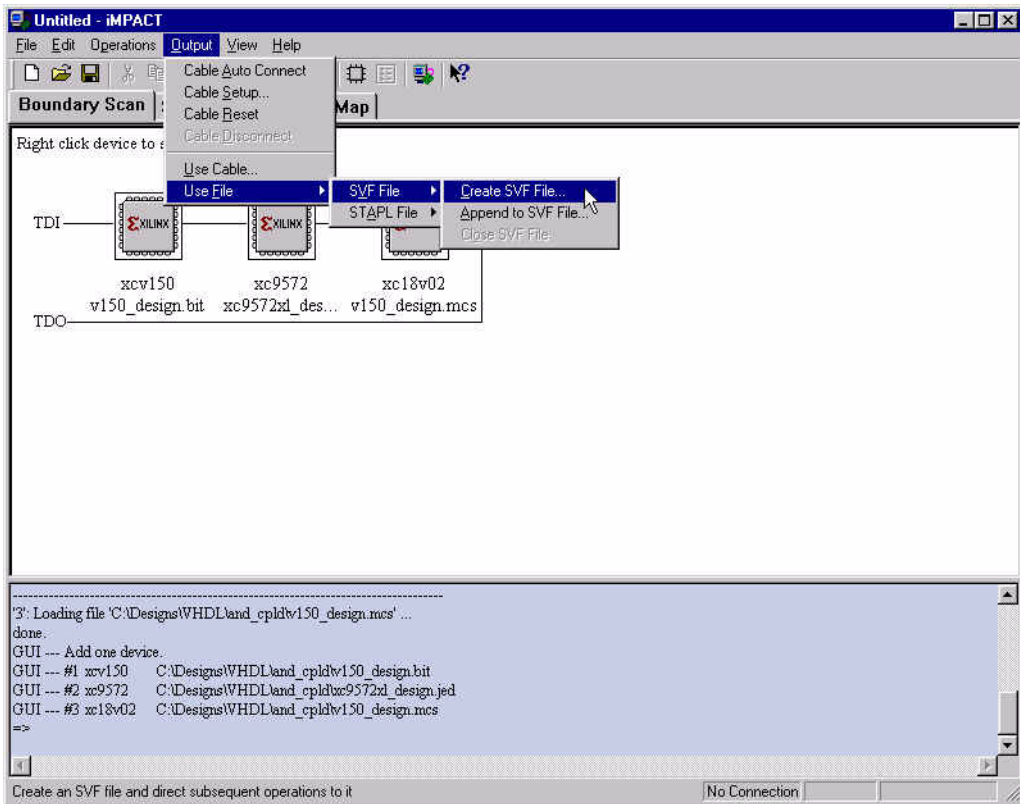


Figure 6-20 Selecting SVF or STAPL File

Writing to the SVF or STAPL File

The process for writing to an SVF or STAPL file is identical to performing Boundary Scan operations with a cable. You simply right-click on a device and select an operation. For instance, in [Figure 6-21](#), you right-click on the first device in the chain and then left-click on Get Device ID. The instructions that are necessary to perform a Get Device ID operation are then written to the file. [Figure 6-22](#) shows what the SVF file looks like after the Get Device ID operation is performed.



Figure 6-21 Selecting a Boundary Scan Operation

```

TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
// Validating chain...
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 21 TDI (1fffff) SMASK (1fffff) TDO (010101) MASK (03e3e3) ;
TIR 0 ;
HIR 16 TDI (ffff) SMASK (0000) ;
HDR 2 TDI (00) SMASK (00) ;
TDR 0 ;
//Loading device with 'idcode' instruction.
SIR 5 TDI (09) SMASK (1f) ;
SDR 32 TDI (00000000) SMASK (00000000) TDO (f0618093) MASK (0fffffff) ;
//Loading device with 'idcode' instruction.
SIR 5 TDI (09) ;
SDR 32 TDI (00000000) TDO (f0618093) ;
//Loading device with 'bypass' instruction.
SIR 5 TDI (1f) ;
TIR 0 ;
HIR 0 ;
TDR 0 ;
HDR 0 ;
SIR 21 TDI (1fffff) SMASK (1fffff) ;
SDR 3 TDI (00) SMASK (07) ;

```

Figure 6-22 SVF File that Gets a Device ID from the First Device in the Chain

Any number of operations can be written to an SVF or STAPL file. For instance, after selecting Get Device ID for the first device in the chain, you can select the second device in the chain and select the Program option. The instructions and configuration data needed to Program the second device are added to the file. After all the desired operations have been performed, select **Output → Use File → SVF File** or **STAPL File → Close SVF File** or **Close STAPL File**. These selections close the file so that no more information can be written to it. To add other operations in the future, you can select **Output → Use File → SVF File** or **STAPL File → Append to SVF File** or **Append to STAPL File**.

Slave Serial Configuration Mode

Slave Serial Configuration mode allows you to program a single Xilinx device or a serial chain of Xilinx devices. To use the Slave Serial Configuration Mode, click the Slave Serial Tab at the top of the iMPACT window and establish a cable connection.

Adding a Device

To add a device, right-click on the iMPACT window and select Add Device (see [Figure 6-23](#)).

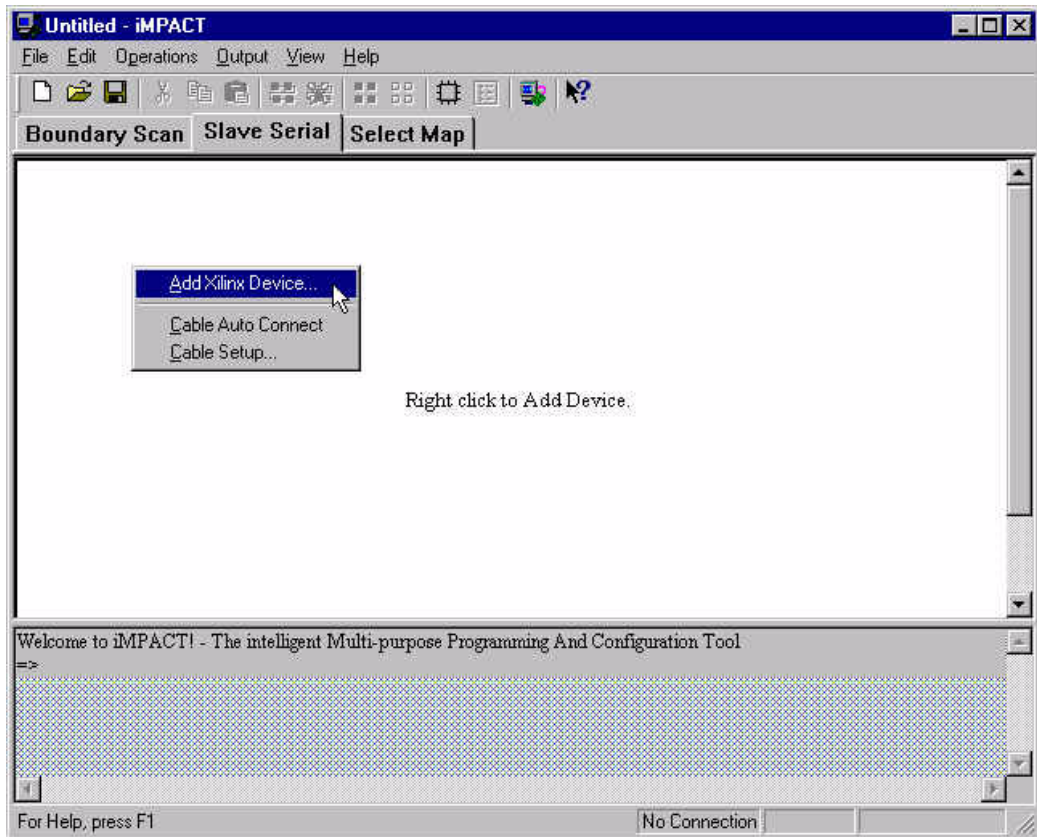


Figure 6-23 Adding a Xilinx Device in Slave Serial Mode

After clicking on Add Xilinx Device, a window appears that allows you to browse to the desired file (see [Figure 6-24](#)).

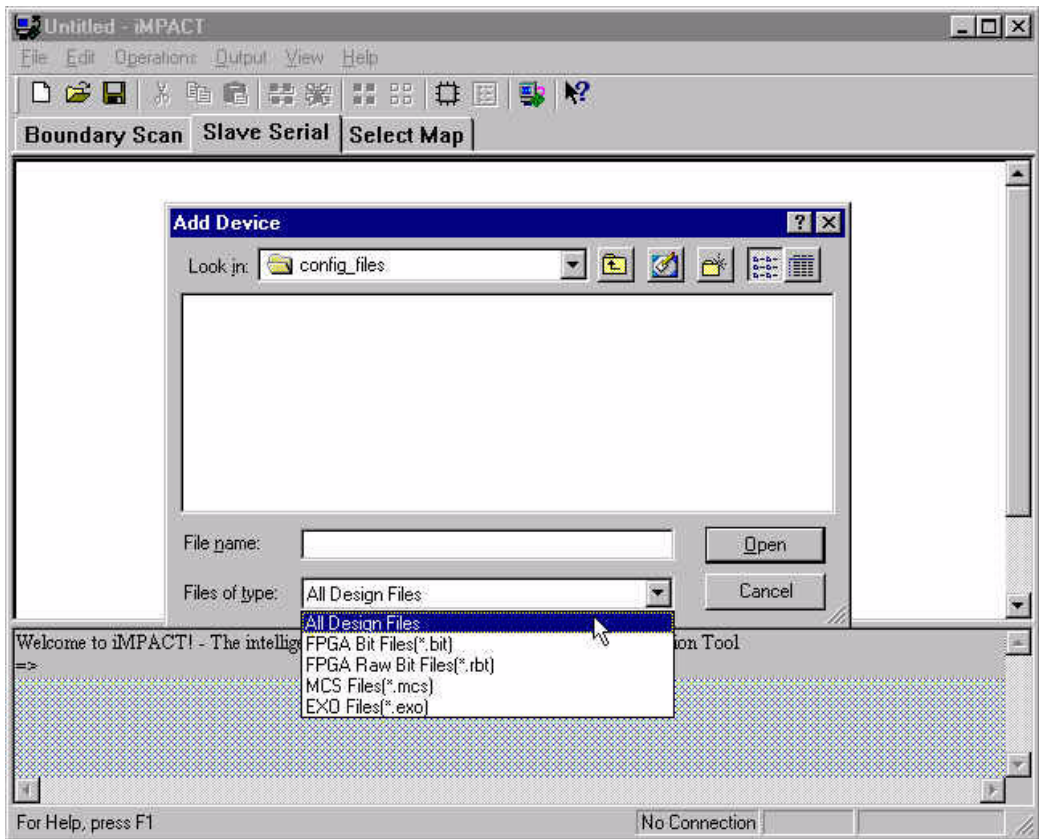


Figure 6-24 Add Device File Types for Slave Serial

Notice in [Figure 6-24](#) that there are a number of different file types to select from. An explanation of each type is below:

- **FPGA Bit File (*.bit)** — Standard bit file created by Bitgen. A Bit file is used to program single FPGA.
- **FPGA Raw Bit Files (*.rbt)** — A Raw Bit File is an ASCII version of the Bit file. The only other difference is that the header information in a Bit File is removed from the Raw Bit File. This file is also created by Bitgen and is used to program a single FPGA.

- MCS and EXO Files (*.mcs, *.exo) — These are PROM files. To program a serial chain of Xilinx devices, a PROM file is used to concatenate all of the Bit files. This single PROM file is then loaded into iMPACT and is used to configure the chain. The PROM files are created by PROM FILE FORMATTER.

Only one file can be loaded for Slave Serial Configuration Mode and so the PROM file must be used for programming a serial chain of devices. Once the desired file is selected, a window similar to [Figure 6-25](#) displays.

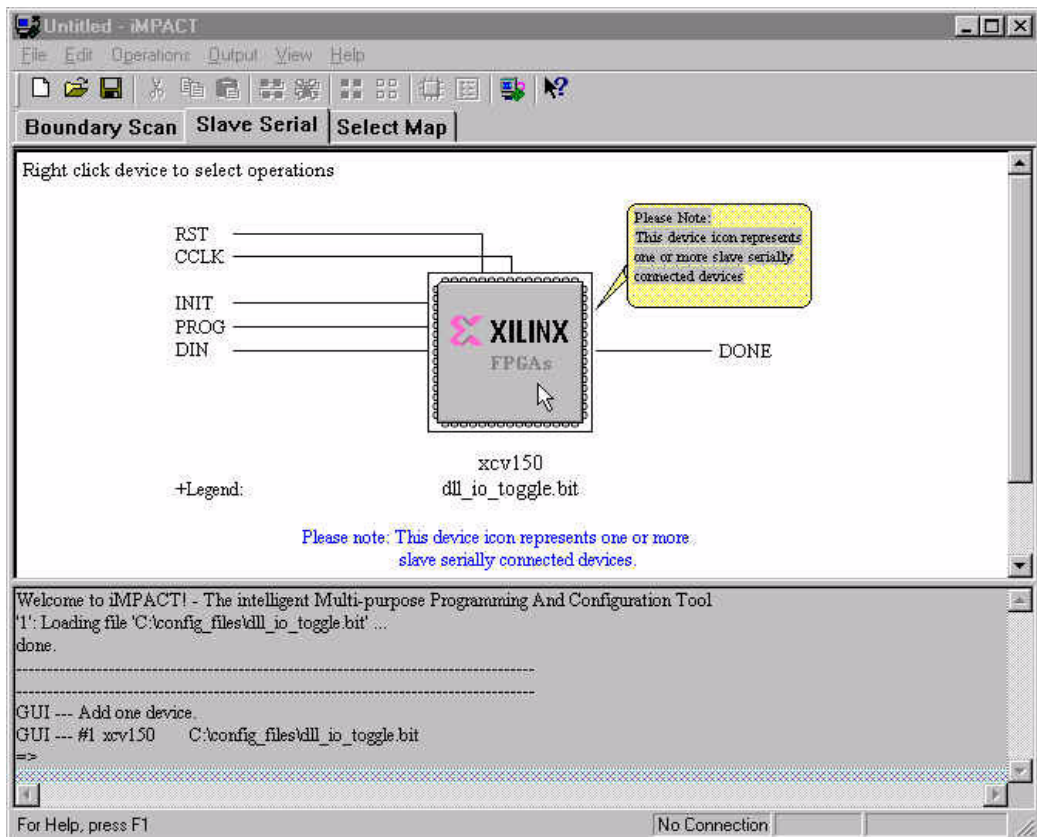
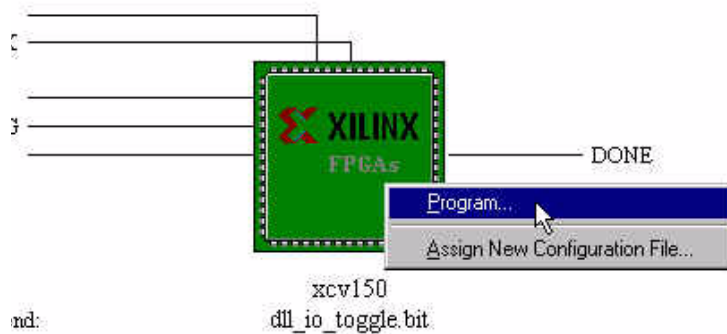


Figure 6-25 Device Loaded with a Single Bit File

Even if a chain of devices is being programmed, only the single device shown in [Figure 6-25](#) displays. This device represents a single device, when a BIT or RBT file is used and represents a chain of devices when a PROM file is used.

Programming the Device

To program a device, right-click on the device and then select Program (see [Figure 6-26](#)).



Please note: This device icon represents one or more slave serially connected devices.

Figure 6-26 Selecting the Program Option

When Program is selected, iMPACT begins programming the device or chain of devices and when it completes, the large message shows that Programming Succeeded (see [Figure 6-27](#)).

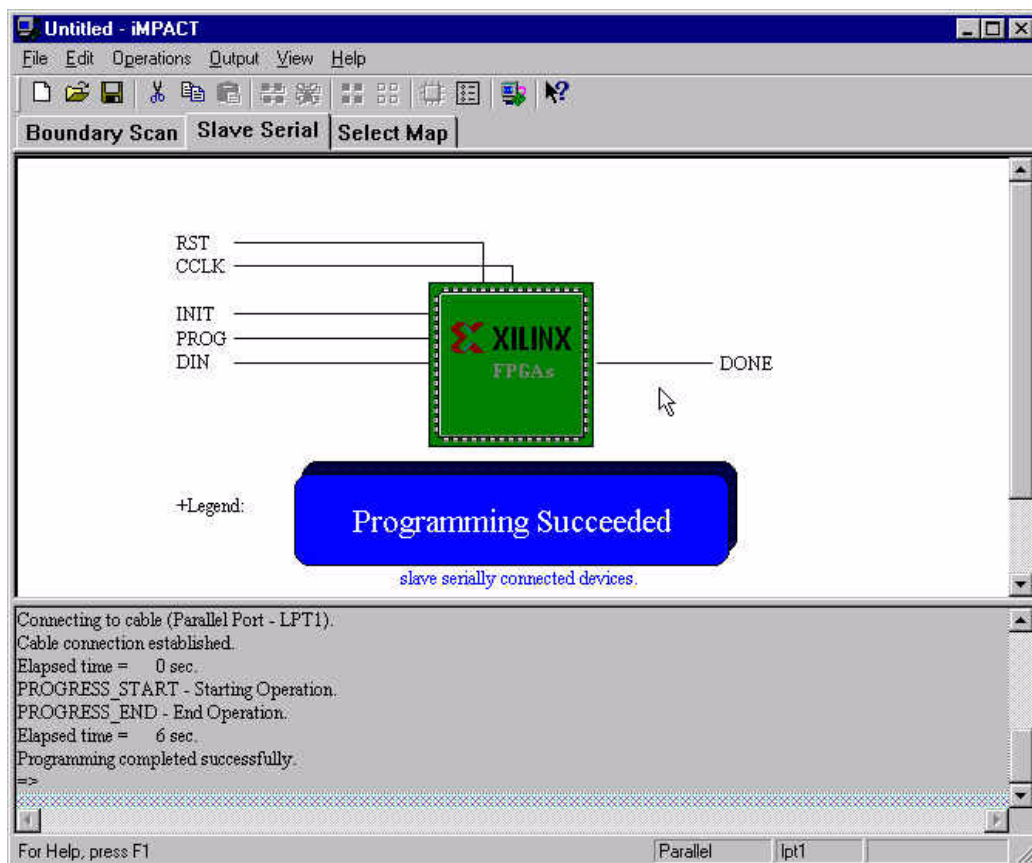


Figure 6-27 Window Shown When Slave Serial Programming Completed Successfully

Troubleshooting Slave Serial Configuration

If configuration fails, the window in [Figure 6-28](#) displays. Notice that the error message appears in the Log Window.

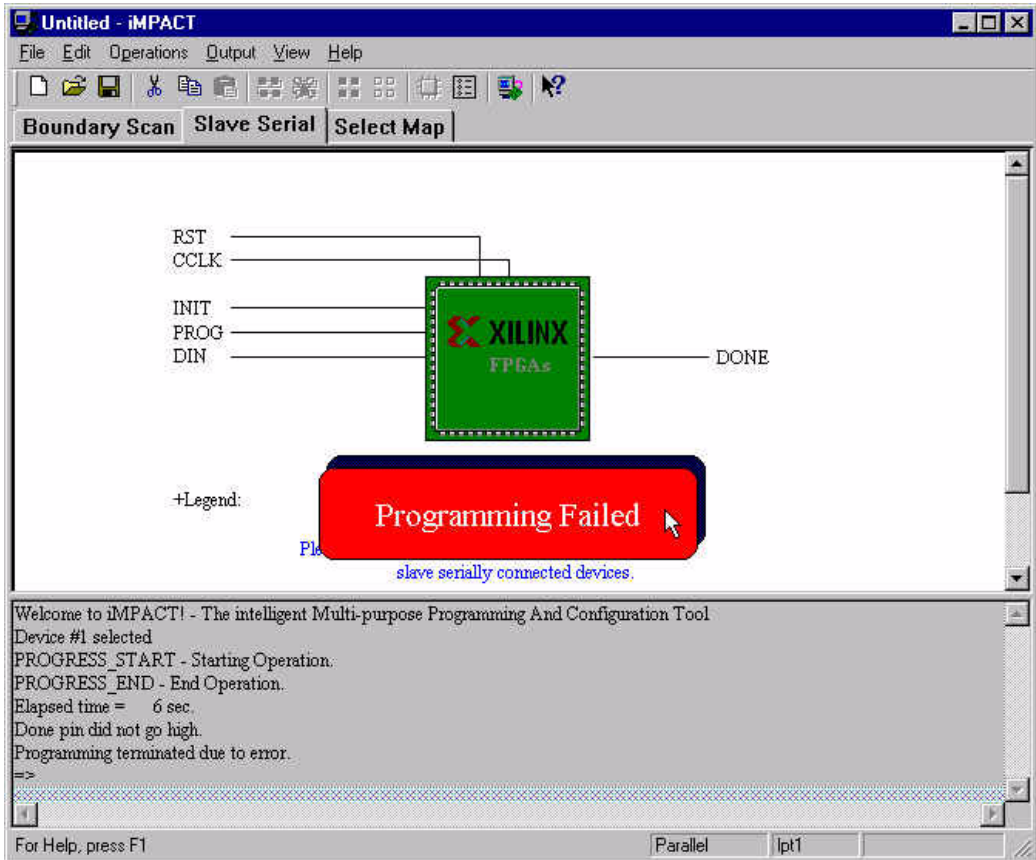


Figure 6-28 Window that Appears when Programming Fails

There are two main error messages that you may encounter. The first is:

DONE pin did not go low. Please check cable connection.

Programming terminated due to error.

The first step in programming through slave serial is that PROG is pulsed low, which erases the device and forces DONE to go low. If the preceding error message appears, it is likely that the PROG or DONE pin of the cable is not properly connected to the device.

The second common error message is:

```
Done pin did not go high.
```

```
Programming terminated due to error.
```

This error message can occur for many reasons. Below are some of the common causes:

- The Mode pins on the device are not set to Slave Serial
- DIN, INIT, or CCLK are not connected.
- Noise is corrupting CCLK or DIN signals.
- The hardware is not set up properly.
- The wrong configuration file was used or the PROM file does not have the BIT files concatenated in the correct order.

If these suggestions do not help resolve the problem, try using the Configuration Problem Solver at <http://support.xilinx.com/support/troubleshoot/psolvers.htm>.

Select MAP Configuration Mode

With iIMPACT, Select MAP Configuration mode allows you to program up to three Xilinx devices. The devices are programmed one at a time and are selected by the assertion of the correct CS pin. To use the Select MAP Configuration Mode, click the Select MAP tab at the top of the iIMPACT window and establish a cable connection. Only the Multilink cable can be used for Select MAP Configuration.

Adding a Device

To add a device, right-click the iMPACT window and select Add Xilinx Device (see [Figure 6-29](#)).

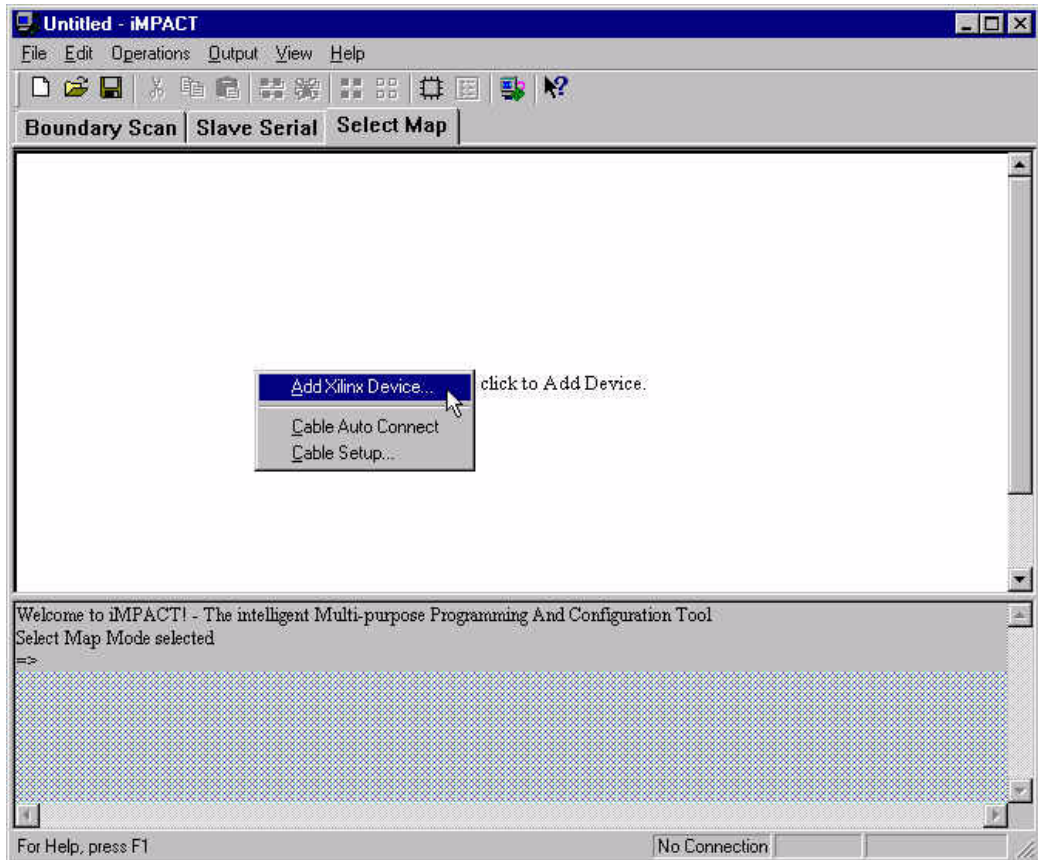


Figure 6-29 Adding a Device for Select Map Configuration

After clicking on Add Xilinx Device, a window display allows you to browse to the configuration (see [Figure 6-30](#)).

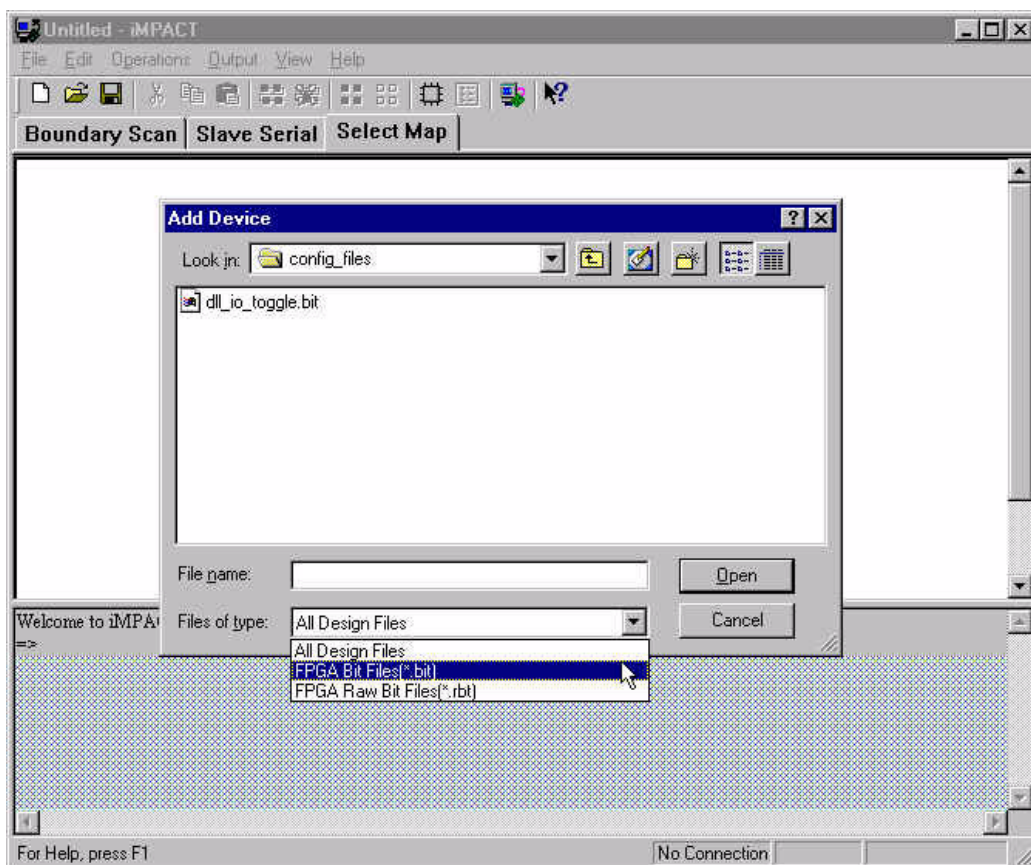


Figure 6-30 Configuration File Types for Select MAP Configuration

Notice in [Figure 6-30](#) that only two file types can be used, FPGA Bit Files and FPGA Raw Bit Files. For a description of these files, see the [“Adding a Device”](#) section.

Once a BIT or RBT file is selected the device displays in the iMPACT window. Up to three devices can be added. [Figure 6-31](#) shows an example where two devices have been added. Notice that each one has a different Chip Select (CS) pin. These correspond to the CS pins on the Multilink Cable. Make sure that the correct CS pin is connected to the correct device. The CS pins can be swapped by dragging and dropping the pins in the window.

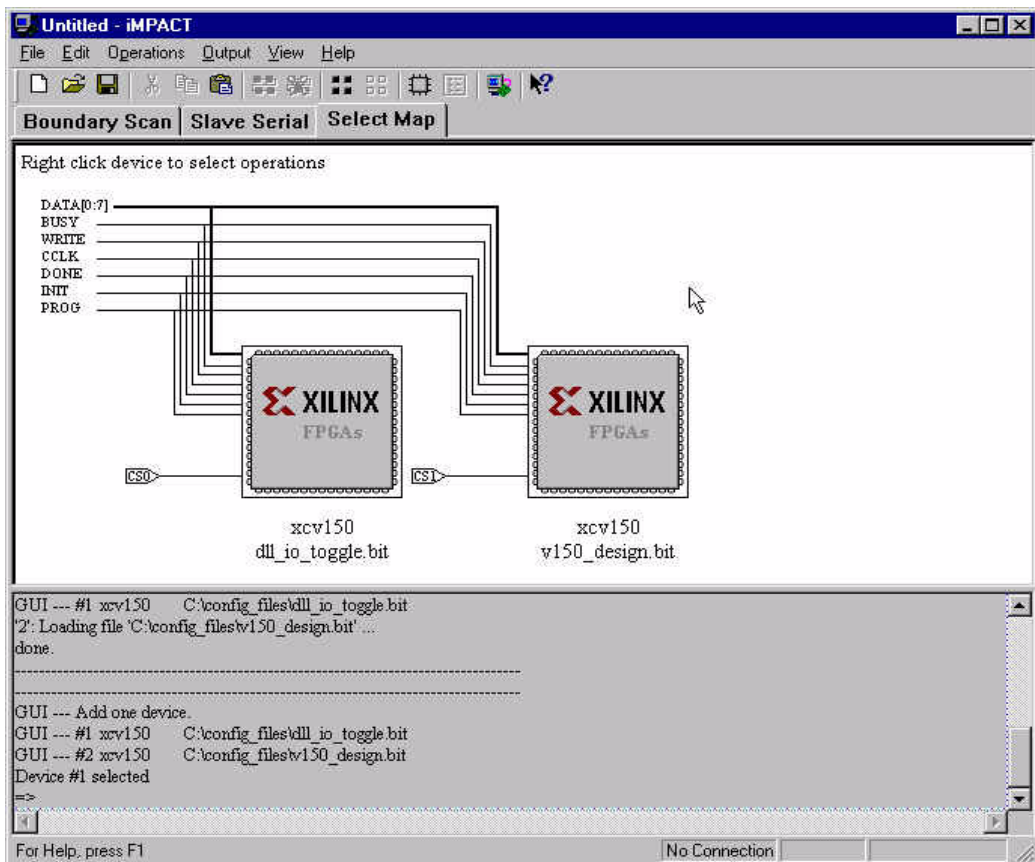


Figure 6-31 Two Devices Added For Select Map Configuration

Programming and Verifying a Device

To program or verify a device, right-click the device and then select Program or Verify (see [Figure 6-32](#)). The Multilink cable asserts the correct CS pin and then performs that operation on that device.

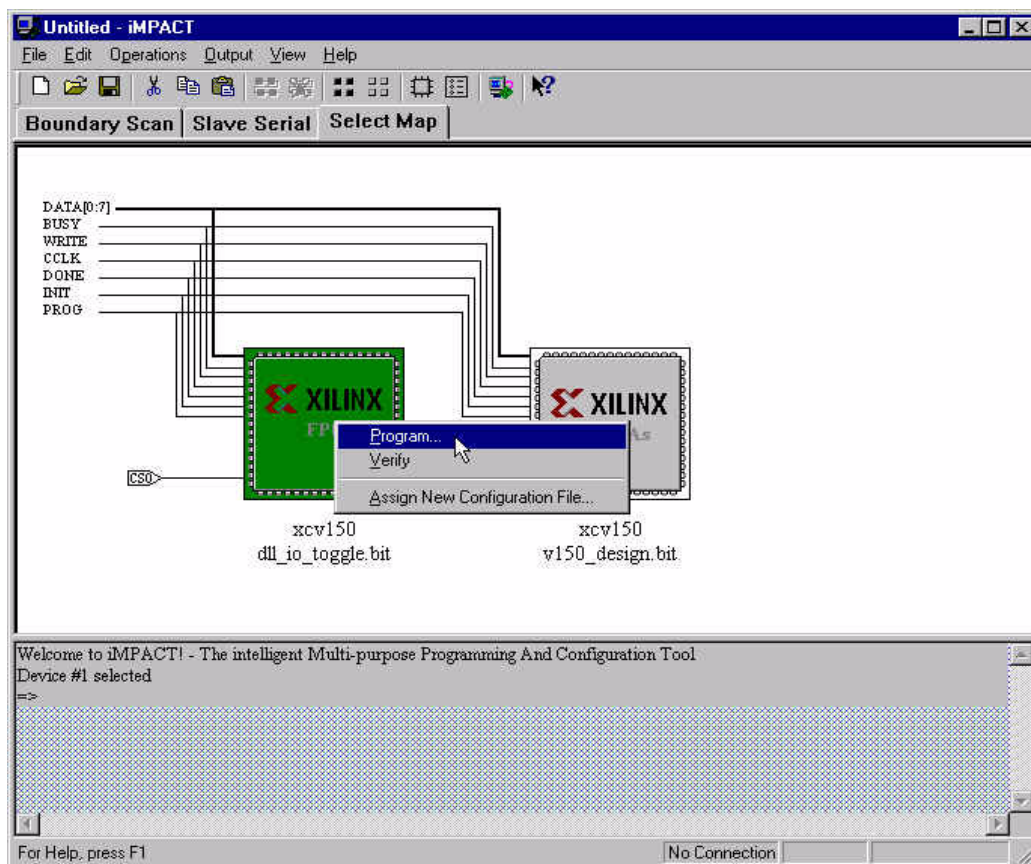


Figure 6-32 Selecting a Program or Verify in Select Map Mode

When Program or Verify is selected, iMPACT performs the operation and a large status message indicates that the operation completed successfully. [Figure 6-33](#) shows a successful Verify operation.

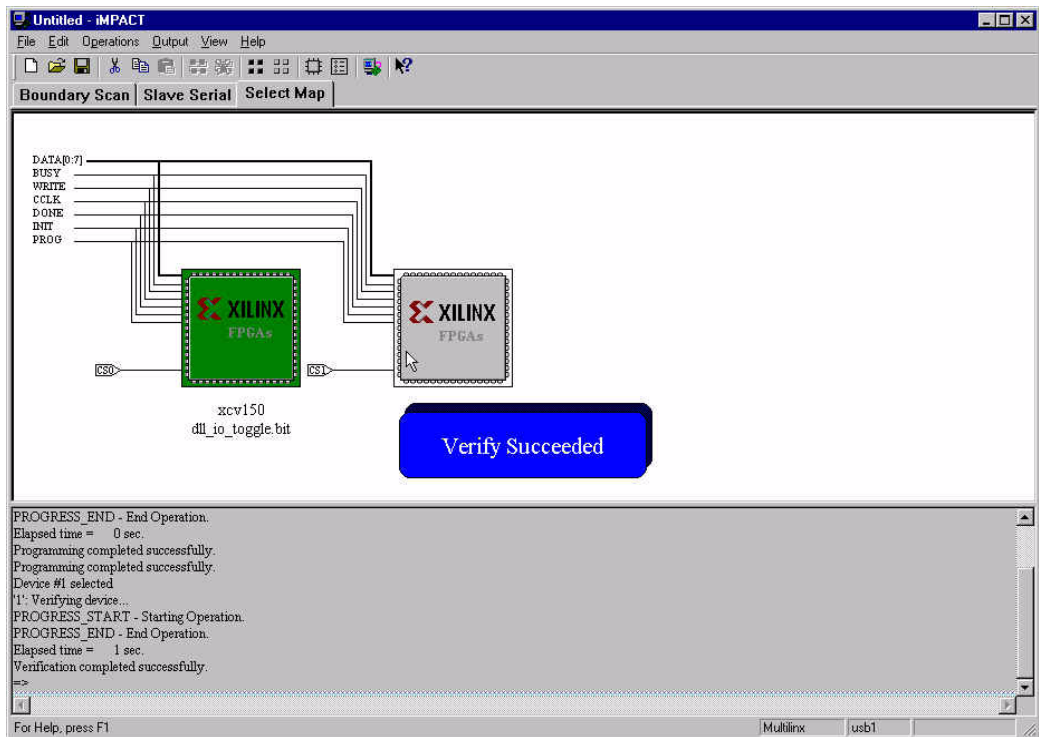


Figure 6-33 Window Shown When Select MAP Verify Completed Successfully

Troubleshooting Select MAP Programming and Verify

If Programming or Verify fails, a large red status message indicates that the operation failed. [Figure 6-34](#) shows a failed Program Operation.

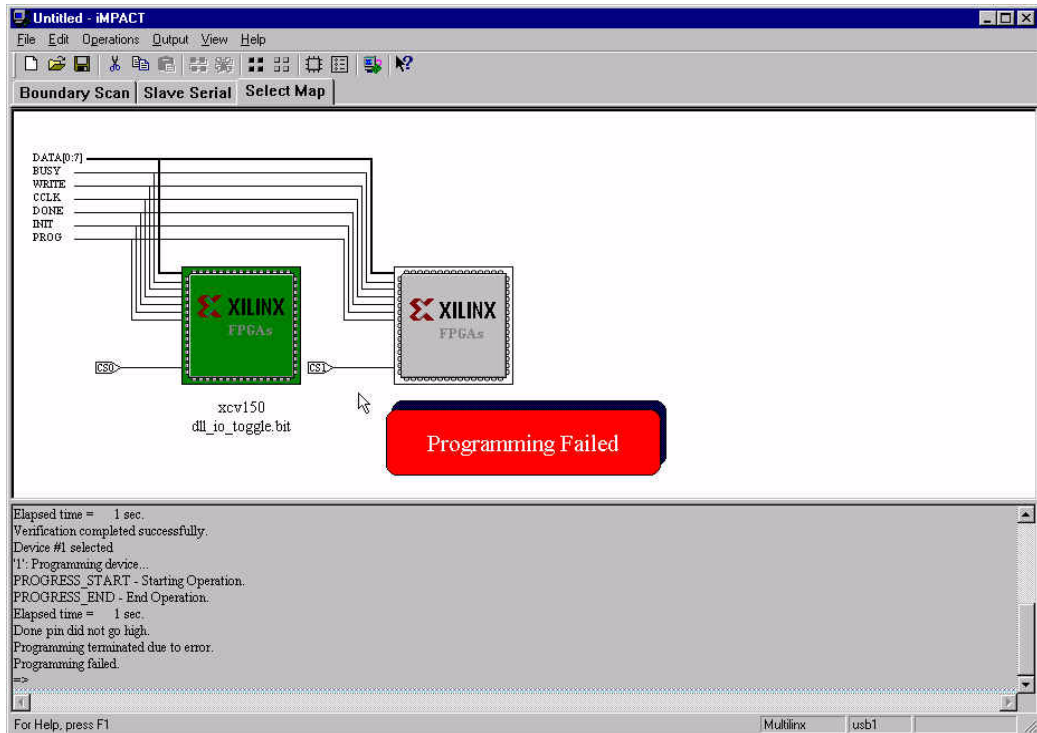


Figure 6-34 Window Shown When Select Map Programming Failed

When Programming fails, the error message will likely read:

```
Done pin did not go high.  
Programming terminated due to error.  
Programming failed.
```

This error message can occur for many reasons. Below are some of the common causes:

- The Mode pins on the device are not set to Select Map Mode.
- One or more of the Select Map signals are not connected properly.
- The wrong CS pin is connected to the device.
- Noise is corrupting CCLK or the DATA lines.
- The hardware is not set up properly.
- The wrong configuration file was applied to the device.

When Verify fails, the error message will likely read:

```
ERROR:Bitstream:98 - There are ## differences.  
ERROR:iMPACT:395 - The number of difference is ##  
Verification failed.
```

The preceding error message can be caused by any of the conditions listed above for a failed Program operation. In addition, the problem might be caused because the BIT File was generated incorrectly. If security is set to Level1 or Level2 or if Persist is set to No, verify will fail. Check the Bitgen options and make sure that Security is set to None and Persist is set to YES.

If these suggestions do not help resolve the problem, try using the Configuration Problem Solver at <http://support.xilinx.com/support/troubleshoot/psolvers.htm>.